

8-16-2024

A Plug-and-Produce Connectivity Framework for Manufacturing Systems

Theodros Abraham Tarekegne
University of South Carolina

Follow this and additional works at: <https://scholarcommons.sc.edu/etd>



Part of the [Mechanical Engineering Commons](#)

Recommended Citation

Tarekegne, T. A.(2024). *A Plug-and-Produce Connectivity Framework for Manufacturing Systems*. (Master's thesis). Retrieved from <https://scholarcommons.sc.edu/etd/7821>

This Open Access Thesis is brought to you by Scholar Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact digres@mailbox.sc.edu.

A PLUG-AND-PRODUCE CONNECTIVITY FRAMEWORK FOR MANUFACTURING SYSTEMS

by

Theodros Abraham Tarekegne

Bachelor of Science
University of South Carolina, 2021

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science in

Mechanical Engineering

College of Engineering and Computing

University of South Carolina

2024

Accepted by:

Ramy Harik, Director of Thesis

Thorsten Wuest, Reader

David Rocheleau, Reader

Ann Vail, Dean of the Graduate School

© Copyright by Theodros Tarekegne, 2024
All Rights Reserved.

ACKNOWLEDGEMENTS

I would like to extend my heartfelt appreciation to my Major Professor, Dr. Ramy Harik, for his unwavering support and guidance during my master's study and research. I am deeply grateful to the neXt Future Factories team, with whom I have collaborated over the course of my graduate studies. I would also like to give special thanks to Fadi Kalach, Max Kirkpatrick, and Ibrahim Yousif for their continual guidance and technical assistance. Recognition is also due to the institutions and companies that supported this work, including the South Carolina Research Authority (SCRA), National Science Foundation (NSF), Siemens, IBM, and Yaskawa.

This research receives partial funding from NSF Award 2119654 "RII Track 2 FEC: Enabling Factory to Factory (F2F) Networking for Future Manufacturing" and "Enabling Factory to Factory (F2F) Networking for Future Manufacturing across South Carolina," funded by South Carolina Research Authority. It is important to note that any opinions, findings, and conclusions or recommendations expressed in this material are solely those of the author(s) and do not necessarily represent the views of the sponsors.

ABSTRACT

The Industrial Internet of Things (IIoT) movement has birthed technologies that enable the fusing of physical and digital environments. However, the extensive number of available options for industrial communication and data formatting has remained a barrier to achieving digital knowledge synonymy in industry. There have been previous attempts to establish standards for industrial connectivity and communication, but legacy machines, as well as custom and non-proprietary devices, still face issues communicating data with other manufacturing equipment. To address these challenges, the Industrial Internet of Things initiative has led to the creation of the Plug and Produce (PnP) concept to facilitate the smooth commissioning of industrial systems without the necessity for intensive manual configuration from system integrators. This thesis aims to apply the PnP concept to the Future Factories manufacturing testbed at the University of South Carolina's McNair Aerospace Center to connect the various hardware and software components that interact within the manufacturing system. The architecture of the PnP systems consists of several protocol-agnostic Agent Gateways (AGs) that connect to the various manufacturing systems. These AGs acquire and aggregate the process data from the various data sources and translate them to the OPC UA framework. Utilizing the information modeling features of the OPC UA framework, the AGs can declare the capabilities and characteristics of the

manufacturing systems to IIoT applications. The AG then pushes the process data to a central Ignition Gateway (IG) platform to enable widespread connectivity, data availability, and application development between different manufacturing facilities and systems.

TABLE OF CONTENTS

Acknowledgements.....	iii
Abstract.....	iv
List of Tables	vii
List of Figures	viii
List of Abbreviations	xi
Chapter 1 Introduction	1
Chapter 2 Literature Review	8
Chapter 3 Theory and Development	40
Chapter 4 Implementation and Discussion	72
Chapter 5 Conclusion.....	106
References.....	111

LIST OF TABLES

Table 2.1 IIoT Protocol Stack Layers	13
Table 2.2 Future Directions for PnP	39
Table 3.1 PnP Requirements for Future Factories	42
Table 3.2 OPC UA Framework Specification Types.....	48
Table 5.1 ACMs for PnP Framework Future Work.....	108

LIST OF FIGURES

Figure 2.1 Hybrid IIoT Architecture.....	10
Figure 2.2 IIoT Protocol Stack.....	12
Figure 2.3 Plug and Produce Taxonomy	17
Figure 2.4 Growth Trends of Networking Protocols	35
Figure 3.1 Future Factories Testbed	43
Figure 3.2 DT of the Future Factories Testbed.....	44
Figure 3.3 Design Process of Custom Information Model	52
Figure 3.4 Development Methodology for AGs.....	62
Figure 3.5 Development Methodology for IG	63
Figure 3.6 PnP Architecture for Future Factories	65
Figure 3.7 Sample IIoT Architecture for Brownfield Manufacturing Scenarios	70
Figure 4.1 Robotics Companion Specification	73
Figure 4.2 Define Namespaces in Model.xml File	75
Figure 4.3 FF Namespace Object in Model.xml File.....	75
Figure 4.4 FF Object Types in Model.xml File	76
Figure 4.5 R01 Object Instance in Model.xml File	78
Figure 4.6 Compilation of Model.xml File	78

Figure 4.7 Output of Model.xml Compilation	79
Figure 4.8 FF Information Model Diagram	80
Figure 4.9 RocketTray Object Instance in UaModeler.....	80
Figure 4.10 MVI Information Model Diagram.....	81
Figure 4.11 DI Reference Namespace Server Interface.....	83
Figure 4.12 Robotics Reference Namespace Server Interface.....	83
Figure 4.13 FF Companion Specification Server Interface	84
Figure 4.14 Generate FF Namespaces and Data Types	86
Figure 4.15 R01 Object Node in namespace_ff_generated.c File	87
Figure 4.16 FF Node IDs in ff_nodeids.h File.....	87
Figure 4.17 FF Base Server Code from main.c File	88
Figure 4.18 Generate MVI Namespace.....	89
Figure 4.19 Config_Correct Variable Node in namespace_mvi_generated.c File	89
Figure 4.20 MVI Base Server Code from main.c File	90
Figure 4.21 Parsing FF Data in MQTT Client	91
Figure 4.22 FF Data Assignment in MQTT Client.....	92
Figure 4.23 Callback Function for Q_VFD1_Temperature Variable Node	92
Figure 4.24 Calling beforeRead_Q_VFD1_Temperature Callback Function	93
Figure 4.25 UAExpert Client Connection to FF OPC UA Server.....	94
Figure 4.26 Parsing MVI Data in MQTT Client.....	95
Figure 4.27 Callback Function for Config_Correct Variable Node	95
Figure 4.28 Calling beforeRead_ConfigCorrect Callback Function	95

Figure 4.29 UAExpert Client Connection to MVI OPC UA Server.....	96
Figure 4.30 Ignition Tag Providers for AGs.....	97
Figure 4.31 Ignition OPC UA Connections for AGs.....	97
Figure 4.32 Adding MVI Agent to Tag Provider	98
Figure 4.33 Client Connection to Ignition OPC UA Server	98
Figure 4.34 MQTT Distributor Module Configuration	99
Figure 4.35 MQTT Transmitters.....	100
Figure 4.36 Subscribe to IG MQTT Server	100
Figure 4.37 Configure MQTT Server in MQTT Engine	101
Figure 4.38 Configure Gateway Network Connection	102
Figure 4.39 Windows PC Ignition OPC UA Server	102

LIST OF ABBREVIATIONS

3GPP	3 rd Generation Partnership Project
ACM.....	Automatic Configuration Mechanism
AG.....	Agent Gateway
AMQP.....	Advanced Message Queuing Protocol
API.....	Application Programming Interface
CAGR.....	Compound Annual Growth Rate
CoAP.....	Constrained Application Protocol
CPS.....	Cyber-Physical System
CPU.....	Computer Processing Unit
DI	Device Integration
DMTF.....	Distributed Management Task Force
DNS-SD.....	Domain Name System Service Discovery
DT	Digital Twin
ERP	Enterprise Resource Planning
ETSI	European Telecommunications Standards Institute
F2F	Factory-to-Factory
GDS.....	Global Discovery Services
GUI.....	Graphical User Interface

HTTP.....	Hypertext Transfer Protocol
IEC.....	International Electrotechnical Commission
IEEE.....	Institute of Electrical and Electronics Engineers
IETF.....	Internet Engineering Task Force
IG.....	Ignition Gateway
IIoT.....	Industrial Internet of Things
IIRA.....	Industrial Internet Reference Architecture
IoT.....	Internet of Things
ITU-T.....	International Telecommunication Union Telecommunication Standardization Sector
LDS.....	Local Discovery Services
LLDP.....	Link Layer Discovery Protocol
M2M.....	Machine-to-Machine
mDNS.....	Multicast Domain Name System
ML.....	Machine Learning
MVI.....	Mobile Visual Inspection
MQTT.....	Message Queuing Transport Protocol
NuSMV.....	New Symbolic Model Verifier
OASIS.....	Organization for the Advancement of Structured Information Standards
OCF.....	Open Connectivity Foundation
OMA.....	Open Mobile Alliance
OPC UA.....	Open Platform Communications United Architecture
OSGI.....	Open Service Gateway Initiative
PLC.....	Programmable Logic Controller

PnP	Plug and Produce
QoS.....	Quality of Service
RAMI 4.0.....	Reference Architectural Model Industry 4.0
RTE	Real Time Ethernet
SDN.....	Software-Defined Networking
SGS	Semantic Gateway as Service
SNIA	Storage Networking Industry Association
SoA.....	Service-Oriented Architecture
SSN	Semantic Sensor Network
TSN	Time Sensitive Networking
UML.....	Unified Modeling Language
UPnP	Universal Plug-and-Play
W3C	World Wide Web Consortium
WS-Discovery	Web Services Dynamic Discovery
XMPP.....	Extensible Messaging and Presence Protocol

CHAPTER 1

INTRODUCTION

1.1 MOTIVATION

The emergence of the Industrial Internet of Things (IIoT) has given rise to technologies that facilitate the integration of physical and digital environments. The industrial sector is increasingly incorporating the use of these technologies to take advantage of the vast amounts of data being generated for various applications. However, the vast array of options for industrial communication and data formatting has posed a significant obstacle to achieving a unified digital knowledge framework between industrial systems. Despite previous efforts to establish standards for industrial connectivity and communication, challenges persist in enabling communication between legacy machines, custom devices, and non-proprietary equipment. Moreover, these IIoT technologies are traditionally created for specific use-cases and applications, preventing industrial systems from rapidly responding to changes in the industrial environment, such as the addition of new sensors and equipment. In response to these challenges, the IIoT initiative has introduced the Plug and Produce (PnP) concept, aiming to streamline the commissioning of industrial systems without requiring extensive manual configuration by system integrators. The motivation behind this thesis is to investigate the PnP concept through its

application to the Future Factories manufacturing testbed. This thesis aims to develop a generalized PnP system that enables connectivity between the various hardware and software components that interact within the Future Factories manufacturing system.

1.2 BACKGROUND

This chapter presents a background regarding the technological initiatives that embody the development of PnP systems. We provide an overview of the Internet of Things (IoT) and subsequently the IIoT to provide context to the description of the PnP concept.

1.2.1 INTERNET OF THINGS

The IoT has emerged as an influential force, garnering significant attention across diverse industries [1]. This technological revolution involves the interconnectedness of devices, machines, and sensors, commonly referred to as "things," through standard Internet technologies. The exponential growth of the IoT is evident, with the global market projected to expand from \$662.21 billion in 2023 to a staggering \$3,352.97 billion by 2030, indicating a remarkable Compound Annual Growth Rate (CAGR) of 26.1% [2].

IoT infrastructure spans various application domains and is increasingly being integrated into businesses, enabling both physical and virtual devices to seamlessly interoperate through data networks, all without requiring direct human intervention [3]. The applications of IoT are numerous and diverse, including domains such as smart cities [4], smart homes [5], energy [6], healthcare [7], and intelligent transportation [8], [9]. The interconnectedness facilitated by IoT is not limited to specific sectors but extends to a wide array of applications, contributing to a more interconnected and intelligent world.

In the realm of smart cities, IoT technologies are harnessed to enhance urban living by optimizing resource management, improving public services, and ensuring

sustainability [4]. Similarly, smart homes leverage IoT to create intelligent environments that enhance convenience, security, and energy efficiency [5]. Energy applications strive to improve efficiency in cost and energy usage [6]. Healthcare applications utilize IoT for remote patient monitoring, data-driven diagnostics, and personalized healthcare solutions [7]. Intelligent transportation systems rely on IoT to enhance route optimization, parking, streetlights, and accident prevention/detection [9]. As IoT applications continue to evolve, they present a growing landscape of heterogeneity, complexity, and increasing demands, underscoring the need for standardized technologies and architectures to ensure seamless networking and interoperability across diverse applications and industries.

1.2.2 INDUSTRIAL INTERNET OF THINGS

The IIoT represents a transformative paradigm in the realm of industrial operations, merging the capabilities of Operational Technology (OT) with the advanced connectivity and intelligence of the IoT [10]. At its core, IIoT leverages interconnected devices, machines, sensors, and systems within industrial settings to enhance efficiency, productivity, and overall operational performance. This convergence of physical and digital realms facilitates data-driven decision-making [11], [12], [13] and autonomous process optimization [14], [15], [16], heralding a new era of smart, connected industries.

In the industrial landscape, IIoT acts as a catalyst for digital transformation, fostering connectivity and intelligence across the entire value chain [17], [18]. The deployment of sensors and connected devices in industrial processes enables real-time monitoring, predictive maintenance, and data-driven insights. This interconnected ecosystem brings forth opportunities for enhanced automation and optimization of

industrial processes, and has led to the creation of smart factories capable of adapting to dynamic market demands [19].

The seamless integration of IIoT technologies holds the promise of unlocking unprecedented operational efficiencies, reducing downtime, and enhancing overall asset management [20]. With a strong foundation in the principles of connectivity, data exchange, and intelligent analytics, the IIoT emerges as a cornerstone in the ongoing evolution of industries toward more adaptive, responsive, and interconnected systems. As industries increasingly harness the power of IIoT, the potential for innovation and efficiency gains becomes a driving force in shaping the future of industrial processes and systems.

1.3 PLUG AND PRODUCE

PnP plays a pivotal role in addressing the evolving challenges of modern manufacturing, where product and innovation cycles are becoming increasingly complex, and there is a growing demand for customized and specialized products [21]. In the face of unpredictable market conditions, production plants must exhibit the capability for rapid adaptation to shifts in capacity and changes in production processes. Traditional plant engineering tends to prioritize specific products and operating points, often at the expense of flexibility. Modularization is identified as a solution to enhance the reconfiguration efficiency of production plants, allowing for swift adaptations to varying demands [22], [23]. This is particularly significant as modularization enables the reuse of engineering solutions across multiple customers, optimizing the overall efficiency of the engineering process.

The landscape of manufacturing is witnessing a shift from mass production to mass customization due to the demand for personalized products [24]. The adaptability of manufacturing facilities to accommodate changes in product specifications is crucial. Achieving flexibility in manufacturing systems relies heavily on minimizing the configuration efforts required to commission and restructure production facilities. Currently, automated production facilities face challenges due to proprietary automation technologies and communication protocols, leading to inflexibility. The PnP concept augments the ability of the system to detect newly plugged devices and configure them efficiently and automatically to perform specific production tasks.

The dynamic nature of today's manufacturing environment is marked by shorter product life cycles and changing market conditions, and therefore necessitates the agility of manufacturing systems to adapt to dynamically changing business needs [25]. However, modifying existing production systems involves time-consuming and manual steps that can be error-prone, leading to costly production interruptions. Control systems in modern manufacturing encompass a range of components, including embedded controllers, sensors, actuators, servers, workstations, and cloud services [26]. The software supporting these systems can be extensive, with intricate architectures designed to meet challenging non-functional requirements such as performance, reliability, and security [27].

The vision of PnP revolves around maximizing the economic sustainability of production systems through three primary initiatives [28]. First, it focuses on enabling PnP capabilities for automation equipment, robots, and machines. Second, it aims to facilitate horizontal and vertical communication between all hardware and software entities, fostering the innovation of new business functions. Finally, it strives to enable seamless

interaction with high-level Manufacturing Execution System (MES) and Enterprise Resource Management (ERP) systems that are easily extendable and adaptable for the introduction of new product specifications, work orders, and equipment. The PnP vision anticipates seamless deployment, optimization, and changeover management strategies, aligning with the overarching goal of enhancing manufacturing flexibility and efficiency.

1.4 THESIS STRUCTURE

Chapter 1 introduces the IoT, IIoT, and the PnP concepts. Subsequently, the structure of the thesis is described.

Chapter 2 includes a literature review of PnP in the IIoT. The survey provides a brief overview of the general IIoT architecture and conducts an investigation on connectivity in the IIoT. The survey describes the taxonomy of PnP and similar concepts in relation to the IIoT. The review subsequently identifies the PnP characteristics and elaborates on current research directions for key enabling IIoT technologies for PnP. This is followed by a discussion on the challenges in incorporating IIoT technologies in PnP systems, followed by potential future research directions to address the identified gaps in the literature.

Chapter 3 defines the use-case and requirements for PnP in the Future Factories manufacturing testbed and provides an overview of the various system components that make up the testbed. The chapter then highlights the capabilities of OPC UA that enable interoperability for PnP systems. The chapter provides a description of the OPC UA framework, including key features that enable interoperability such as the OPC UA *base information model* defined from the OPC UA Core Specification as well as OPC UA *companion specifications*. The chapter goes into detail on the process of developing a

custom information model, allowing for the combination of companion specifications to define the capabilities of assets more emphatically in the manufacturing environment. Subsequently, the chapter describes the role of various modules and software integrations for developing the Ignition Gateway (IG) platform as a scalability solution for PnP systems. The development methodology for the PnP framework is then described. Lastly, the thesis presents a roadmap for implementing the PnP framework for brownfield manufacturing scenarios.

Chapter 4 presents the architecture of the developed PnP framework for the Future Factories manufacturing cell. The chapter describes the approach for developing Agent Gateways (AGs) through the building of custom information models for the Rocket Assembly station and Mobile Visual Inspection station in the Future Factories manufacturing cell. Subsequently, the configuration process for the development of IGs to enable scalability for the PnP framework is presented. The system utilizes various combinations of software packages and libraries to enable Factory-to-Factory (F2F) connectivity and communication while remaining communication protocol agnostic.

Chapter 5 summarizes the contents of the thesis and suggests future work for further development of the PnP system. A description of the situation of research for the Future Factories manufacturing cell is provided, providing an overview of the various prior research works within the Smart Manufacturing domain.

CHAPTER 2

LITERATURE REVIEW

A literature review pertinent to the themes explored in this thesis is presented. First, we introduce the concept of the IIoT emphasizing the architectural and connectivity requirements. Subsequently, we delineate the PnP taxonomy and define the characteristics inherent to PnP with a focus on enabling IIoT technologies and current research directions. Lastly, a discussion is presented on the challenges associated with facilitating PnP systems, accompanied by potential solutions.

2.1 INDUSTRIAL INTERNET OF THINGS

The IoT refers to the interconnected network of physical devices, vehicles, appliances, and other objects embedded with sensors, software, and connectivity, enabling them to collect and exchange data over the internet [29]. The fundamental concept of IoT involves turning everyday objects into smart devices by enabling them to communicate and share information autonomously [30]. The IIoT exists as a subset of the IoT, tailored specifically for industrial applications [10]. The IIoT is closely intertwined with the broader Industry 4.0 concept [31], which is an initiative describing the ongoing transformation of traditional manufacturing and industrial processes through the integration of cutting-edge technologies, intelligent automation, and data-driven insights. The IIoT harnesses the

power of interconnected devices, sensors, and systems to enhance efficiency, productivity, and data-driven decision-making in industrial environments. Unlike traditional industrial systems, the IIoT integrates digital intelligence into physical processes, fostering automation, intelligent decision-making, and the seamless information exchange across the industrial landscape [19]. The IIoT ecosystem enables the concept of Smart Manufacturing [32], where machines, equipment, and production lines operate collaboratively in a dynamic and interconnected environment. The IIoT is characterized by its emphasis on connectivity, data analytics, and the convergence of OT with information technology (IT), paving the way for a new era of intelligent and adaptive industrial operations.

2.1.1 ARCHITECTURE OF THE IIOT

The IIoT is a rapidly expanding field that enables the interconnection of devices and sensors to collect and exchange data over the internet. Reference architectures refer to the various models that serve as a foundation in the design, deployment, and management of IIoT systems by providing a higher level of abstraction to identify issues and challenges across different application scenarios [33]. These architectures are crucial for ensuring the reliability, scalability, and security of IIoT systems [34], impacting industries such as transportation [9], supply chain logistics [17], manufacturing [35], and energy [36]. Designing an effective IIoT architecture requires a focus on specific aspects of the IIoT system, such as extensibility, scalability, modularity, and interoperability among heterogeneous devices using various technologies. Commonly adopted approaches include multilayer descriptions organized around the services offered at each level, tailored to specific technologies, business needs, and technical requirements [33]. The design of an

IIoT architecture depends on factors such as the number and types of devices, data processing requirements, network connectivity, and security needs.

Numerous reference architecture frameworks have emerged in the past, addressing diverse application contexts for the IIoT [37]. This review specifically focuses on two prominent reference frameworks: the Industrial Internet Reference Architecture (IIRA) from the IIoT Consortium and the Reference Architectural Model Industry 4.0 (RAMI 4.0) from Platform Industries 4.0. The IIRA concentrates on the functionality of the industry domain, including business, operations, data analytics, and domain-specific applications [38]. It offers specific viewpoints catering to different stakeholders, providing guidance to system architects in building IIoT systems. These viewpoints cover categories such as *business*, *usage*, *functional*, and *implementation*, each serving as a foundation for designing domain-specific IIoT architectures. The RAMI 4.0 model extends the IIRA model toward the life cycle and value streams of manufacturing applications [18]. Unlike the IIRA, RAMI 4.0 aims to harmonize different user perspectives, offering a common understanding of relations and attributes between individual components for IIoT solutions [33].

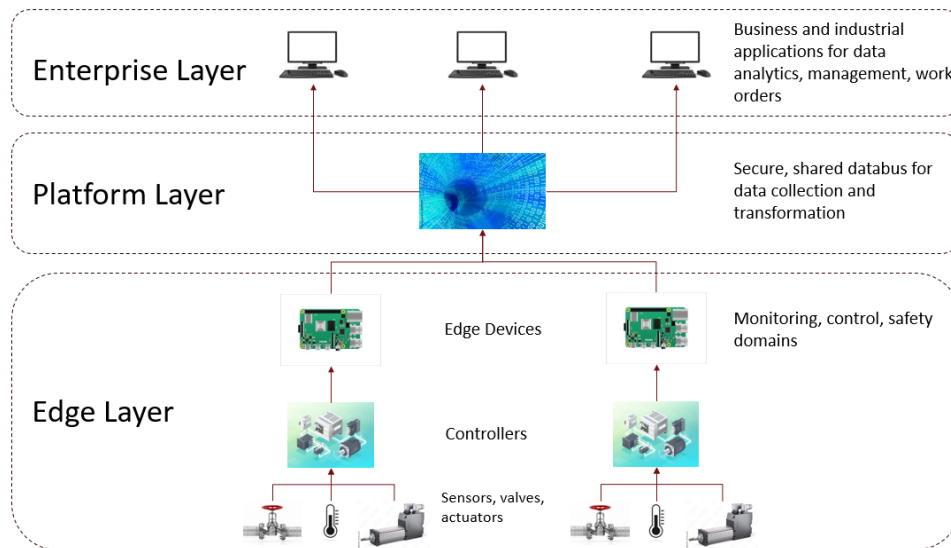


Figure 2.1: Hybrid IIoT Architecture

These reference architectures [18], [37], [39] serve as the basis for designing use-case specific IIoT architectures. These multiple-tier architectures enable the design of dynamic models and services for both business and industrial applications while also addressing the challenges posed by the heterogeneous nature of devices and networks. Hybrid IIoT architectures typically follow a three-tier pattern connecting edge, platform, and enterprise tiers through service networks [33]. Figure 2.1 shows a graphical representation of a generic hybrid IIoT architecture. The edge tier encompasses sensors, controllers, and actuators interconnected by local area networks, connecting to the platform tier through edge gateways. The platform tier, in turn, links to the enterprise tier, implementing domain-specific applications and providing end-user interfaces through service networks.

2.1.2 CONNECTIVITY IN THE IIOT

The IIoT represents a transformative paradigm shift in the industrial landscape, enabled by the seamless integration of advanced sensors, intelligent devices, and data analytics into traditional industrial processes [10]. At the core of this technological revolution lies the crucial element of connectivity, which plays a pivotal role in unlocking the full potential of the IIoT. Connectivity in the IIoT refers to the ability of devices and systems to communicate, share data, and collaborate in real-time across a network infrastructure, ultimately fostering a more responsive, intelligent, and efficient industrial ecosystem [38]. Connectivity is an essential feature for the IIoT because the establishment of a robust and interconnected network enables the seamless exchange of data between various systems within an industrial environment. This interconnectedness allows for real-time monitoring, control, and coordination of industrial processes, leading to improved

operational efficiency and agility [40]. The ability to collect, transmit, and analyze data in real-time empowers organizations to make data-driven decisions, optimize processes, and respond swiftly to changing conditions.

A key objective of enabling IIoT connectivity is to mitigate the use of isolated systems that are dependent on proprietary solutions [33]. The overarching aim is to foster an environment conducive to data sharing and interoperability among existing closed subsystems and forthcoming applications, both within and across diverse industries. IIoT connectivity also serves to provide interoperable communications among network endpoints for the purpose of facilitating system integration [38]. Interoperability can be achieved at various levels of the IIoT Protocol Stack, with solutions ranging from custom integration to Plug and Play interfaces based on open standards. The IIoT Protocol Stack is shown in Figure 2.2.

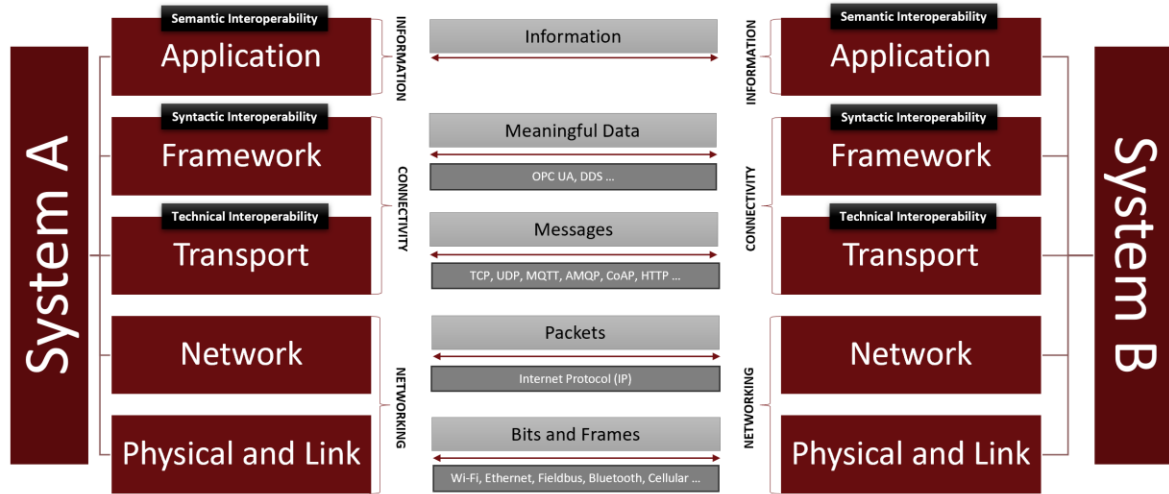


Figure 2.2: IIoT Protocol Stack

The IIoT Protocol Stack consists of a layered architecture that facilitates communication and interoperability among systems in industrial environments. The protocol stack is based on the IIoT Communications Stack Model from [38] and the IIoT

Protocol Stack from [33]. Various models of the IIoT Protocol Stack [31], [33], [34], [38], [41] can differ slightly in that some of the layers are combined or separated according to the application. For example, technologies such as Wi-Fi and Ethernet perform operations simultaneously in the physical and link layers, and therefore are represented as a single layer in the stack, as shown in Figure 2.2. While there is not a single, standardized IIoT protocol stack, several communication technologies are commonly used across different layers. The role of each layer in the IIoT Protocol Stack [38] is described in Table 2.1.

Table 2.1: IIoT Protocol Stack Layers

Physical Layer	Involves the transmission of physical signals, sent as bits, via wired or wireless means to establish connections among participants.
Link Layer	Encompasses the transmission of frames, or series of bits, through signaling protocols over a shared physical link between adjacent connected participants.
Network Layer	Manages the exchange of packets, routing them across multiple links to facilitate communication between non-adjacent participants.
Transport Layer	Facilitates the exchange of messages, which can vary in length, between applications of participating entities.
Framework Layer	Governs the exchange of structured data, including state, events, and streams, offering configurable quality-of-service features for communication among participant applications.
Application Layer	Depends on the data-sharing mechanisms provided by the framework layer to enable communication and collaboration between applications of participating entities.

In the context of the IIoT Protocol Stack, the interoperability concept is categorized into three specific definitions based on the corresponding IIoT layer [38]. Technical interoperability pertains to the capacity to exchange information as bits and bytes, syntactic interoperability involves the ability to employ a shared data structure and a defined set of

rules for exchanging meaningful data [30], [42], and semantic interoperability refers to the capability to interpret the meaning of exchanged data unambiguously within the appropriate context [43]. The actual application itself plays a crucial role in delivering semantic interoperability and requires the prior establishment of technical and syntactical interoperability in the IIoT environment [33].

Connectivity serves as the backbone for the implementation of enabling technologies for the IIoT, such as edge computing [44], [45], cloud services [46], [47], big data analytics [48], [49], data fusion [50], [51], and machine learning [52]. Connectivity also enables the integration of intelligent sensors and actuators into industrial processes, creating a network of smart devices that can communicate and collaborate autonomously [33]. This interconnectedness enhances predictive maintenance capabilities, allowing for the early detection of equipment failures, minimizing downtime, and optimizing maintenance schedules [10]. Furthermore, connectivity supports the concept of Digital Twins (DTs), enabling real-time monitoring and simulation for better decision-making [41].

In summary, IIoT connectivity enables the creation of intelligent, interconnected, and data-driven industrial ecosystems. The seamless flow of information across the IIoT network empowers industries to enhance efficiency, reduce costs, and embrace innovative technologies, driving the ongoing evolution of industrial processes in the digital age.

2.2 PLUG AND PRODUCE

In this section, the taxonomy of PnP and similar concepts is defined. The characteristics inherent to PnP are described, and the current research directions of IIoT technologies for PnP are introduced.

2.2.1 PLUG AND PRODUCE TAXONOMY

The concepts of "Plug and Play," "Plug and Work," and "Plug and Produce" play interconnected roles in the realms of industrial automation, IoT, and smart manufacturing. The IIoT has been classified as the intersection between the IoT and CPSs according to [33]. "Plug and Play" originated as a concept in the computer domain, where peripherals are automatically detected and configured by the operating system [53]. This concept has been extensively applied across the IoT domain [54], [55], [56], [57], [58], [59], facilitating communication between various devices to observe and interact with the surrounding environment. A specific term has not been created to define the intersection between Plug and Play and the IoT, as this domain is simply referred to as Plug and Play IoT. However, applying the Plug and Play concept to CPS systems has resulted in the formation of the "Plug and Work" concept.

"Plug and Work" introduces a layer of complexity by requiring technical foundations to determine the information exchange during the startup of software and hardware components in a production environment. This concept, integral to industrial standards like AutomationML and OPC UA, emphasizes the secure and confidential transfer of sensitive information between industrial machines [60]. Plug and Work mechanisms ensure that the exchange of information adheres to defined standards within the industrial environment while simultaneously meeting the demands of end-users [61].

The literature suggests that the application of the "Plug and Play" concept to the IIoT has resulted in the formation of the "Plug and Produce" concept. "Plug and Produce" aligns closely with the familiar Plug and Play concept employed in IT systems. In the context of industrial automation and the IIoT, Plug and Produce refers to systems that can

communicate effectively without extensive human intervention [62]. This concept is pivotal for the future smart factory, where machines, production lines, and storage systems collaborate within a network of CPSs. Plug and Produce is closely associated with the IIoT, emphasizing the utilization of standardized communication protocols, data representation formats, information translation modalities, and the ability of devices to self-configure and self-optimize [20], [63]. In the IIoT landscape, it facilitates heterogeneous systems' communication both horizontally and vertically, enabling autonomous interaction between disparate systems, such as digital supply chain networks and industrial production equipment [17]. In this manner, the Plug and Produce concept enables disparate systems to exchange information, trigger actions, and control each other.

Together, these concepts together form a comprehensive framework for seamless integration, communication, and operation within both industrial and commercial ecosystems. While both the “Plug and Produce” and “Plug and Work” concepts share the goal of simplifying the integration of devices in CPSs, Plug and Produce tends to emphasize open-communication between heterogeneous systems, protocol agnosticism, and system adaptability, whereas Plug and Work focuses more on establishing straightforward integration processes based on developed standard communication interfaces. Based on these definitions, we develop the taxonomy for Plug and Produce systems in relation to the associated intersecting research domains in Figure 2.3.

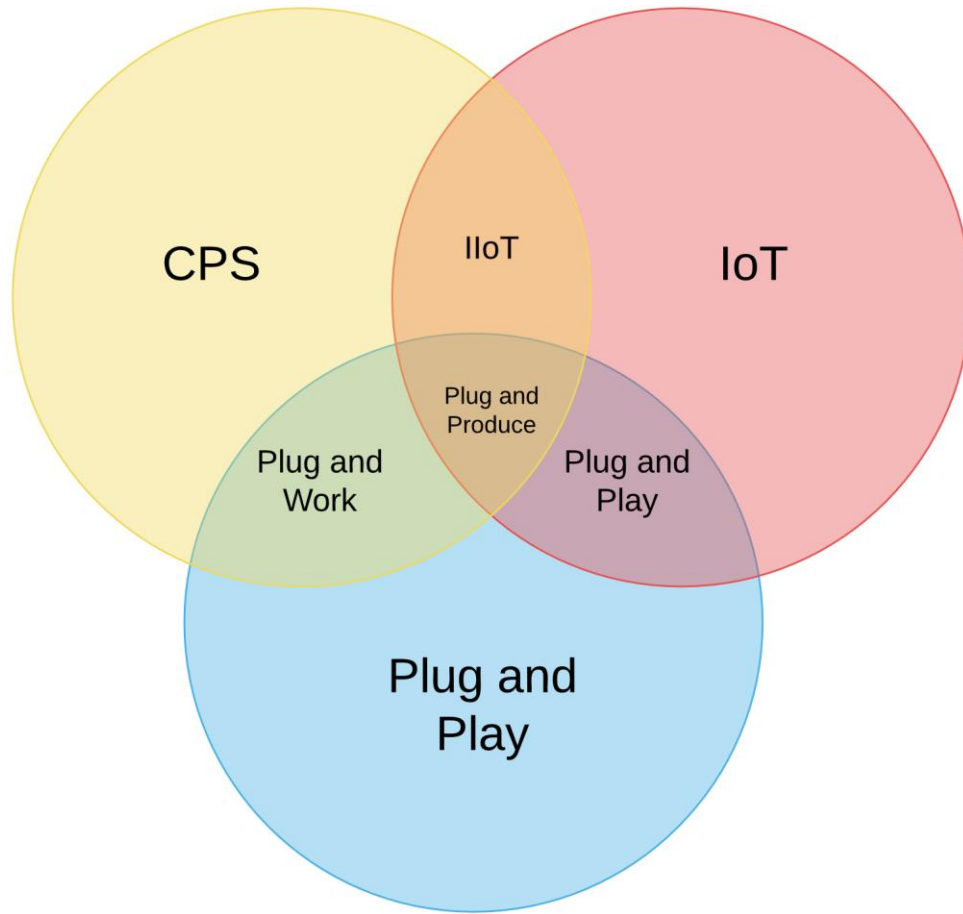


Figure 2.3: Plug and Produce Taxonomy

2.2.2 PLUG AND PRODUCE CHARACTERISTICS AND CURRENT RESEARCH DIRECTIONS

This review investigates the primary characteristics of PnP systems related to IIoT connectivity, which are defined to be interoperability, scalability, and mobility. Security is also an essential characteristic of PnP systems but falls outside the scope of this review, and as such is not covered. We introduce enabling technologies for these PnP characteristics and elaborate on current research directions for these technologies in PnP systems. While the requirements of these technologies may intersect with the other PnP characteristics, these technologies are categorized according to the purpose behind their implementation in PnP systems. This review focuses on generalized and widespread

enabling technologies that can be applied to various use-cases within manufacturing, as there exists a vast array of application-specific IIoT technologies that can be applied to PnP systems but fall outside of this scope.

2.2.2.1 INTEROPERABILITY

As manufacturing systems evolve, diverse devices and systems require the capacity to communicate and collaborate amongst each other. In the context of PnP systems, interoperability refers to the seamless exchange of information and functionalities between various components, both within and across different industrial domains [64]. This harmonious integration fosters a dynamic and interconnected ecosystem where machines, sensors, and controllers can work in tandem to accomplish tasks autonomously [65]. Research regarding enabling seamless IIoT connectivity has yielded a plethora of standards aimed at achieving universal interoperability between interconnected components. By emphasizing standardized communication, PnP systems transcend the barriers posed by disparate devices, ensuring a unified connectivity framework that enables the achievement of the overarching goals of an industrial automation system. Given the plethora of existing communication standards, translation mechanisms are also being utilized to support system interoperability [66], [67], [68]. The ability to translate communication protocols ensures that devices, each adhering to their specific protocol, can effectively communicate and comprehend information.

2.2.2.1.1 STANDARDIZATION OF IIOT PROTOCOLS

The standardization effort in the IIoT plays a pivotal role in the advancement of PnP systems, catering to the specific requirements of commissioning processes [69]. This is particularly crucial in the current push towards Cyber-Physical Systems (CPS), where

the harmonization of information models becomes a critical factor [28]. The IIoT is heavily contingent on the standardization of communication technologies, encompassing aspects such as interoperability, usability, trustworthiness, and ensuring uninterrupted business operations. Leading networking initiatives such as IEEE 802.15.4a and IETF are central to this standardization initiative [64]. Presently, various industrial consortia, organizations (IEEE, WC3, OASIS), and IoT management frameworks (ITU-T, oneM2M, OCF) are driving the standardization efforts for numerous IIoT technologies [70]. Additionally, domain-specific standardized bodies and institutes are working to address diverse industrial challenges related to connectivity (e.g., 3GPP, IETF6Lo, IEC, OSGi, ETSI DASH7) and interoperability (e.g., IEEE PLC, IPv6 Forum, OMA, oneM2M, DMTF, SNIA).

Interoperable connectivity currently poses a significant challenge for industries due to the amalgamation of heterogeneous devices, diverse network architectures, and complex distributed environments [64]. The standardization of IIoT protocols plays a crucial role in establishing an interoperable connectivity within an IIoT ecosystem. These protocols empower IIoT devices to observe, listen, comprehend, and execute tasks by enabling seamless communication, data exchange, and decision-making [71]. A primary objective of IIoT systems is to construct a device-friendly protocol stack that facilitates the interoperable communication of devices, fostering knowledge exchange and the accomplishment of their intended objectives [64]. Notably, at the network level, communication standards (WirelessHART, WIA-AP, and ISA100.11a) are extensively employed for the development of industrial interfaces, even though challenges related to reliability and scalability persist in such models. At the transport level, research initiatives

[68], [72] emphasize the importance of the standardization of various transport protocols, such as TCP/IP and HTTP(S), alongside diverse encoding formats like JSON, BIN, and XML. Regarding OPC UA, the standardization effort extends to interaction patterns with services [73], encompassing actions such as read and write, to facilitate user interaction with the data. At the framework level, information models excel in providing contextualized data; however, the lack of common rules or structures across heterogeneous models poses a challenge in establishing uniformity in information representation between various systems. At the application level, achieving semantic interoperability involves defining OPC UA information models based on desired companion specifications [68].

In the assembly domain, the establishment of standards has become indispensable for creating adaptable systems that ensure compatibility among assembly equipment modules [74]. Importantly, the collaborative nature of standardization efforts extends beyond individual companies or organizations, emphasizing a mutually beneficial approach. This collaborative effort is grounded in domain knowledge maturity, ensuring not only interoperability and integration but also wider acceptance of the technology. Together, these concerted standardization efforts are essential steps toward realizing seamless communication and compatibility in the evolving landscape of smart and interconnected systems.

2.2.2.1.2 INFORMATION TRANSLATION

In the pursuit of achieving interoperability within the IIoT, substantial challenges persist despite ongoing standardization efforts. The need for a semantic IIoT architecture that accommodates multiple IIoT protocols is evident [75]. Specifically, at the transport

layer, the absence of interoperability remains a significant barrier to effective communication among IIoT devices.

Derhamy et al.'s work [76] identified a gap in the investigation of OPC UA interoperability within a multi-protocol setting, leading to the proposition of an OPC UA translator. This translator aims to be compatible with various protocols such as CoAP, HTTP, and MQTT by employing mapping techniques to an intermediate format [66]. To achieve uniformity between information models, a mapping must be established between a subset of OPC UA services and an intermediate format, as well as between the OPC UA address space and IIoT protocol message structure. The method involves mapping OPC UA to an intermediate format that, in turn, can be mapped to other standard communication protocols like CoAP, HTTP, and MQTT. While the protocol translator lacks semantic translation capabilities, it is suggested that an individual semantic translator could complement its functionality.

Desai et al. [75] propose the Semantic Gateway as Service (SGS) concept to serve as a bridge between sink nodes and IIoT services. The gateway, utilizing CoAP, XMPP, or MQTT protocols, semantically annotates data using Semantic Sensor Network (SSN) ontology. The SGS facilitates interoperability at both the messaging protocol and data modeling levels, connecting low-level raw sensor information with knowledge-centric application services.

Taking a different approach, Katti [77] emphasizes the necessity of a mechanism to translate semantic models to a payload received by the OPC UA client, ensuring compatibility with the expected format in the OPC UA server (lowering transformation). Conversely, service output represented in the OPC UA information model must also

undergo transformation to align with the corresponding semantic model (lifting transformation).

Pauker [78] introduces an innovative approach for generating OPC UA information models by automatically transforming Unified Modeling Language (UML) class diagrams, which is a standard language for modeling software systems. While some model elements can be mapped between UML and OPC UA, challenges arise due to the inherent incompatibility of certain UML concepts and model elements with the OPC UA framework.

In a different context, Koo [79] proposes the concept of an IoT device-name-system architecture as an interoperability-enabling technology. This architecture facilitates the analysis and translation of an IoT device's identification system and resource request format, allowing resource requests between heterogeneous IoT platforms. Notably, the DNS architecture is tested on a microcomputer, indicating its suitability for low-power IoT applications. These diverse approaches and proposals contribute to the ongoing discourse on addressing interoperability challenges in the evolving landscape of the IIoT.

2.2.2.2 SCALABILITY

In the IIoT, scalability refers to the ability of a system to handle increasing workloads effectively without compromising the performance, efficiency, and quality of the system. With the rise of smart manufacturing and interconnected systems, PnP systems require the capability to seamlessly integrate new technologies, devices, and components into existing manufacturing infrastructures, and should be able to easily accommodate the addition of new sensors, actuators, and other IIoT devices [62]. The scalability of PnP systems can also be enhanced through the efficient distribution of computing resources to

meet varying production demands. Scalability is therefore defined to be a critical characteristic in PnP systems, and with the advent of the IIoT, the capability to support an ever-increasing quantity of IIoT devices and process large volumes of data is more necessary than ever before.

2.2.2.2.1 OPC UA

The scalability of OPC UA within PnP is highlighted by its ability to accommodate varying scopes of functions and support diverse platforms through OPC UA servers [80]. Basic communication is fundamental aspect to realizing the Plug-and-Produce environment, requiring automated device discovery capabilities and self-description, along with interoperability across different systems [81]. In this context, OPC UA's service-based machine-to-machine communication stands out for its adherence to standardized invocations and concepts for semantic description. The scalability extends not only to the technical aspects of communication but also to the organizational level, as OPC UA enables interoperability across different systems, facilitating seamless integration of devices and systems from various vendors.

Liu's examination of OPC UA's messaging capabilities compares client-server and PubSub interaction patterns to examine the protocol's scalability [82]. The PubSub communication pattern introduces benefits to system scalability, as it is especially useful in manufacturing scenarios where multiple clients are involved. The distinction between broker-based and broker-less modes provides flexibility, allowing for optimization based on specific network and system requirements. The efficiency of PubSub mode, particularly in reducing client processing loads during data changes, further supports its scalability in the context of IIoT environments with potentially numerous connected devices.

Drahos et. al. [83] emphasize the role of OPC UA in messaging over global networks to support various PubSub scenarios. The definition of mappings on messaging protocols such as Advanced Message Queuing Protocol (AMQP) and MQTT enhances OPC UA's scalability, offering a secure and efficient means of data sharing across connected devices. The ability to operate over cloud-based networks adds a layer of scalability crucial for modern industrial applications that span geographically distributed environments.

Burger et al.'s [84] investigation into resource management capabilities concerning potential bottlenecks and Computer Processing Unit (CPU) utilization in OPC UA communication contributes valuable insights into the protocol's scalability. Despite negligible impacts on memory and network resources, identifying the CPU as the primary bottleneck highlights the need for optimization in server-side processing. This insight is crucial for addressing scalability concerns and ensuring the efficient operation of OPC UA in diverse industrial settings.

OPC UA stands as a linchpin technology in ensuring scalability within the PnP paradigm. Its flexibility, standardized communication, and support for diverse functionalities make it an essential element in the advancement of interconnected and scalable industrial systems. The protocol's adaptability to different communication patterns and its emphasis on secure, standardized data exchange position OPC UA as a fundamental technology for the evolving landscape of industrial automation.

2.2.2.2.2 CLOUD COMPUTING

Cloud computing has played a pivotal role in enhancing the scalability for PnP systems. In the foundation of every IIoT platform lies a network of sensors or “things” that

provide information about the surrounding environment. The management of data from these elements is traditionally handled within cloud computing systems [58]. Traditional on-premise computing platforms, while suitable for certain tasks, face limitations in processing large amounts of data in real-time. Additionally, scaling on-premise computing platforms to the level required for training data-driven machine learning models proves to be expensive and challenging. Cloud computing platforms, characterized by their capability to process large volumes of data efficiently, present an ideal solution for these demands in the PnP landscape.

The combination of IIoT technologies with cloud computing has emerged as a powerful paradigm [85]. Middleware serves as a crucial interface between software and physical objects, enabling communication among heterogeneous devices. Leveraging the open connectivity and computing environments of the IIoT complements the capabilities of cloud-computing infrastructures, offering features such as virtualization, scalability, lifecycle management, and multi-tenancy.

The benefits of cloud computing platforms can also be realized when used in tandem with CPS architectures. In a cloud-based CPS architectures, features such as DT monitoring and control, Big Data, and graphical application modules are used to facilitate custom development of dashboards and data analytics for factories [86]. By applying cloud computing architectures to a distributed control environment, PnP systems can realize intelligent control at the network edge. The translation of various communication modalities of industrial machines to the standard OPC UA protocol ensures compatibility between cloud computing platforms and industrial systems. Additionally, applications on the cloud that receive the industrial data are often run in Docker containers to offer

enhanced performance and scalability. Beno et. al. [87] exemplify the integration of OPC UA servers with cloud computing platforms like Microsoft Azure. In this experiment, an OPC UA server, an intermediary edge device, and the Azure Cloud are interconnected. The edge device serves as the central point of communication between the local offline network and the internet, facilitating the seamless transfer of OPC UA data to the Azure Cloud, where it can be interacted with in diverse ways. This integration showcases the practical application of cloud computing for processing and analyzing data from PnP systems, exemplifying the scalability and versatility it brings to industrial environments.

2.2.2.2.3 EDGE COMPUTING

Edge computing plays a vital role in advancing the scalability and efficiency of PnP systems, contributing to the optimization of industrial processes [88]. Unlike cloud-centric IIoT architectures, where data travels to a centralized cloud server for processing, edge computing involves performing computational tasks physically closer to the data source, at the edge of the network. By deploying computational capabilities at the edge of the network, critical decisions can be made in real-time instead of relying on centralized cloud processing [89]. This local processing not only reduces latency but also ensures that devices can respond swiftly to changing conditions or requirements [44].

Edge computing also serves to facilitate autonomous device collaboration in PnP systems. Localized processing enables devices to communicate, coordinate, and collaborate without relying on continuous communication with a central server [90]. This autonomy enhances the efficiency of PnP systems by allowing devices to operate in a coordinated manner without constant external supervision. As the number of connected devices and sensors increases, edge nodes can distribute the processing load efficiently,

preventing congestion and optimizing resource utilization [91]. This scalability is essential for accommodating the growing complexity of industrial ecosystems.

Furthermore, edge computing can also be used to complement the capabilities of cloud computing infrastructures through cooperation mechanisms to enable intelligent manufacturing [45]. By decentralizing computational power from central servers to the network periphery, edge computing is better able to support cloud computing by handling tasks that require real-time processing while delegating big data analytics and complex optimization problems to the central cloud server. Edge devices also act as intermediary nodes that collect heterogeneous data and translate to a common standard [92]. Such standards allow manufacturers to define semantic models that contextualize heterogeneous data, providing a higher level of perception to PnP to enable autonomous and intelligent decision making.

2.2.2.2.4 TASK OFFLOADING

In the rapidly growing IIoT ecosystem, the surge in connected devices has led to a substantial increase in data generation. This data is often processed and analyzed in the cloud, which can result in elevated network traffic and latency in environments where thousands of computation tasks are to be performed in parallel. On the other hand, devices at the edge of the network are traditionally resource-constrained, and therefore may lack the capability to compute large volumes of data. To address these challenges, task offloading techniques can be employed, which involves the transfer of data processing tasks from the cloud to the edge of the network, as well as from the edge of the network to the cloud, to find the most optimal endpoint to carry out the task. This approach enhances

the scalability of IoT services, and IIoT services by extension, by reducing latency, improving network efficiency, and enabling real-time data analysis.

Task offloading entails partitioning computationally exhaustive resources in mobile applications and utilizing cloud resources for computations [93]. A task offloading framework defines the rules for identification of computationally exhaustive resources from code snippets and controls the delegation of these IoT tasks to the most optimal processing device, which can either be local to the device or in the cloud. A large challenge of implementing task offloading in IIoT environments is defining how to consistently optimize delegation of IoT tasks for a dynamic environment. Implementing computation offloading in IIoT environments requires an adaptive framework capable of consistently optimizing the delegation of tasks in dynamic conditions.

Task offloading frameworks often leverage machine learning (ML) algorithms to accomplish this task, as highlighted in [94], which provides a comprehensive comparative analysis of different ML technologies for task offloading. These ML algorithms are defined to designate which tasks are computationally exhaustive, and which can be processed by the device at the edge of the network. Extensive studies on task offloading techniques have been researched, emphasizing the necessity of performance enhancement of IIoT environments. However, the criteria upon which tasks are delegated according to must be defined according to the requirements of the use-case.

Aljanabi et. al. [95] stress the critical necessity of task offloading techniques in network administration to realize benefits such as balancing network traffic load, saving energy in low-power devices, and improving Quality of Service (QoS) metrics related to latency in real-time applications. This approach considered load balancing and delay

parameters to select the best edge-node for each task and to determine the optimal offload policy. Similarly, Sun [96] incorporates a task offloading scheme focusing on optimizing service accuracy in addition to the power-delay trade-off, demonstrated through a case study for IIoT edge intelligence. Using similar task offloading criteria to [95], [96], Zhao [16] defines a three-hierarchical offloading optimization strategy in IIoT networks, using relays, computational access points, and an optimization process for latency and energy consumption reduction. The system performance is measured by analyzing a linear combination form of the latency and energy consumption of the system to identify computationally exhaustive tasks.

Mai et. al. [91] develop an in-network computing paradigm to offload application-specific tasks from end-hosts to network devices with higher processing capacities. In contrast to [16], [95], [96], where the goal is to identify and subsequently offload computationally exhaustive tasks, the criticality of IoT tasks is instead the primary criteria to motivate decision-making. The task criticality is evaluated by a complex event processing tool which identifies meaningful complex events by analyzing, filtering, and matching semantically low-level simple events from multiple sources in real-time, upon which lightweight critical tasks are offloaded to the INC devices.

Rather than using a cloud-based or edge-based architecture, Ghosh [97] proposes a task offloading solution using terminal-to-terminal networks, relying on direct communication between end-users without an infrastructure backbone. A new symbolic model verifier is used to verify three prediction-based offloading schemes that exploit mobility patterns and temporal contacts of nodes to predict future data transfer opportunities.

Considering a much more holistic approach, Aazam et. al. [98] present a taxonomy of task offloading schemes and considers a wide variety of factors for evaluating task offloading decision criteria. The criteria used to determine the task offloading sequence consisted of detection of excessive computation or constrained resources, reducing latency to meet specification, load balancing, permanent or long-term storage requirements, data management and organization, privacy and security, and data accessibility, and affordability, feasibility, and maintenance.

2.2.2.3 MOBILITY

The essence of PnP lies in the seamless integration and interoperability of various devices and components within an industrial environment. In this context, mobility refers to the ability of devices and components to easily connect to and transfer between industrial systems without significant manual intervention. Therefore, mobility is defined to be a key characteristic of PnP systems as it enables the systems' flexibility, adaptability, and portability in the dynamic industrial landscape. In this section, we define several key technologies that enhance the mobility of PnP systems.

2.2.2.3.1 SERVICE ORIENTED ARCHITECTURES

Service-Oriented Architectures (SoAs) play a crucial role in enhancing the mobility of PnP systems by providing a distributed and collaborative framework for the IIoT. Unlike traditional automation systems, which are generally hierarchical and centralized, SoAs focus on distributed systems that consist of collaborating assets with encapsulated information describing their functionalities [69]. These assets offer services that can be easily utilized by other participants in the system, allowing for a more adaptable approach

to solving tasks. SoA services can then be combined to solve different system tasks; this process is referred to as orchestration.

The concept of service orchestration is integral to enabling the functions of CPSs [99]. In this context, services are defined as functions that impact the virtual world of an asset, encapsulated in a way that external users can access them through virtual interfaces. This encapsulation ensures that the internal workings of the service remain a black-box, known only by its inputs and outputs. The autonomy and independence of services contribute to the adaptability, expandability, and stateless operability of the overall CPS, allowing it to efficiently handle diverse tasks.

PnP extensively employs SoA at the application level, identifying autonomous and self-contained components as services accessible to other services or applications through the public exposition of Application Programming Interfaces (APIs) [62]. Auto-configuration mechanisms within service-oriented platforms are focused on discovery mechanisms, aiding the identification of previously registered services in a service directory. Additionally, the presence of a service composition mechanism provides a list of discoverable services that each service must be aware of before utilization.

By leveraging SoA principles, PnP systems achieve adaptability, expandability, and efficient operability. The decentralized nature of encapsulated services enables solutions to be provided independently by relevant system elements, ensuring that the complexity of the program remains manageable and economic. Overall, SoAs provide the architectural foundation for mobile and agile PnP systems.

2.2.2.3.2 AUTOMATIC CONFIGURATION MECHANISMS

Automatic configuration mechanisms (ACMs) play a pivotal role in the PnP paradigm in enhancing mobility of industrial systems. Commissioning such systems traditionally involves a labor-intensive and expensive process of manually installing, configuring, and integrating a multitude of sensors, actuators, and controllers [26]. Over the last 15 years, efforts in the realm of the PnP initiative has sought to automate commissioning, with a historical focus on network discovery and proprietary technologies.

Network protocols are typically defined in some way that, after some basic configuration, neighboring devices can automatically begin the negotiation process to identify and communicate with each other. However, to truly reach an autonomous end-to-end communication service within and between layers in the IIoT architecture, IIoT environments require that several network protocols be used in combination with each other in an intelligent manner [1]. Widespread modern technologies such as Universal Plug-and-Play (UPnP), commonly used in enterprise/home network scenarios, fall short of meeting industrial requirements due to the inherent security risks associated with the automatic authentication that occurs between UPnP-enabled devices. This emphasizes the need for specialized ACMs to enable autonomous configuration in industrial environments.

The introduction of ACMs in industrial environments, as highlighted in [100], is paramount for streamlining the development, deployment, and maintenance of industrial systems. The choice of ACM significantly influences the mobility of the system, enabling manufacturers to seamlessly add or remove devices without manual configuration, regardless of the device vendor.

The evolution of ACMs in industrial systems has historically been rooted in the automatic detection of field devices by a Programmable Logic Controller (PLC) on specific fieldbuses, progressing from serial communication interfaces to current Real Time Ethernet (RTE) interfaces [62]. Different proposals explore the use of lower layers of TCP/IP protocols to auto-configure industrial devices into the RTE network. For instance, Imtiaz et al. [101] suggest an approach involving auto-assignment of MAC addresses combined with a Link Layer Discovery Protocol (LLDP) at Layer-2 to discover device locations in network topology. Innovative approaches such as an ad-hoc channel, as proposed in [102], coexist with RTE channels for automatic identification of industrial devices using Web Service Dynamic Discovery (WS-Discovery) based on SoA.

Duerkop et al. [25] modify this approach by substituting WS-Discovery with OPC UA as a discovery mechanism to identify industrial devices in the network. OPC UA introduces two discovery services for identifying other OPC UA instances within a network: Local Discovery Services (LDS) which pinpoints instances within the same subnet, and Global Discovery Services (GDS) which extends this capability across diverse subnets. By allowing the automatic configuration and deployment of OPC UA servers solely based on information from industrial devices, OPC UA systems gain the capability to become first-class PnP systems through enhancing their mobility.

The semantically described information model from the OPC UA server can also be combined with Software Defined Networking (SDN) features to enhance the automatic configuration and deployment of OPC UA [63]. An augmented version of LDS, referred to as LDS with Multicast Extension (LDS-ME), utilizes Multicast Domain Name System (mDNS) broadcast messages carrying Domain Name System Service Discovery (DNS-

SD) information to discern OPC UA instances. These broadcast messages traverse the entire network, ensuring every OPC UA LDS-ME receives the information to enable the addition of new OPC UA devices to the network without the necessity of preconfiguring the OPC UA counterparts.

Koo [79] introduces the concept of an IIoT device-name-system (DNS) architecture, showcasing its potential for low-power IIoT applications. This architecture facilitates resource requests between heterogeneous IIoT platforms and offers reconfiguration options based on specific application needs.

While many protocols inherently include capabilities to enable automatic device discovery, manual orchestration is typically still required by ACMs to enable end-to-end communication services in multi-protocol industrial communication networks. Nevertheless, the incorporation of ACMs contributes significantly to the mobility and efficiency of industrial systems to meet the demands of PnP.

2.2.2.3.3 WIRELESS NETWORKING

Wireless networking has emerged as a crucial enabler for enhancing the mobility of PnP systems, as the flexibility required by IIoT communications are typically addressed using wireless links [33]. The rapid advancement of wireless technology over the past decade has provided a diverse array of options for configuring the communication layer of industrial applications. The availability of these various options ensures their continued relevance to the dynamic needs of industry.

Exploring these options, Drahos et. al.'s comparison of network protocols reveals a notable shift towards wireless networking in industrial connectivity trends [83]. While traditional fieldbuses and Industrial Ethernet have been historically dominant, wireless

networks are clearly emerging as the fastest-growing segment. This shift underscores the increasing importance of wireless technologies in M2M communication in industry. The growth of these industrial networking modalities are showcased in Figure 2.3.

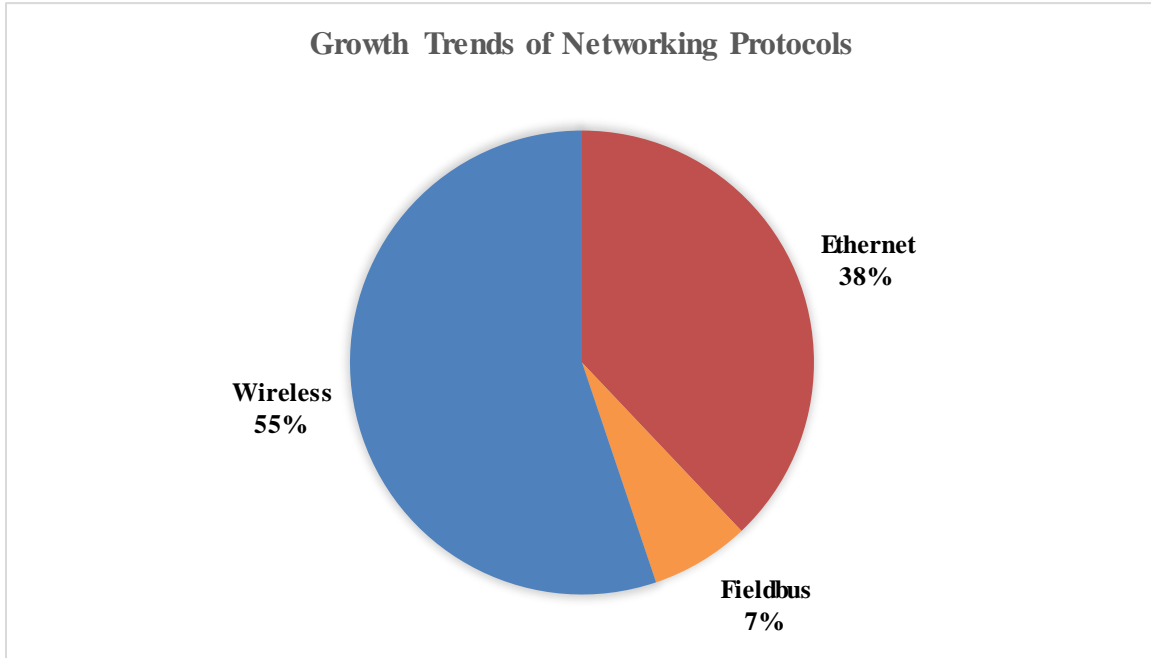


Figure 2.4: Growth Trends of Networking Protocols [83]

Saqlain et al. [103] discussion on wireless sensor networks (WSNs) emphasizes their effectiveness in monitoring industrial machine activities, offering increased flexibility. WSNs enable collaborative interactions with multiple sensor devices to achieve common objectives, a feature that aligns seamlessly with the dynamic requirements of PnP systems. Various wireless technologies, including RFID, Wi-Fi, Bluetooth, Wi-Fi direct, 4G LTE, Z-wave, and ZigBee, are employed to transport heavy data traffic, providing capabilities such as guaranteed system latency and high bandwidth support. PnP ontologies can also be merged with WSNs to improve the support of automatic network configuration for Time Sensitive Networking (TSN) applications [104].

Notably, 5G technology is also gaining prominence in industrial environments, offering advantages such as flexibility, mobility, productivity, quality, safety, sustainability, and utilization [105]. Barring et. al. [106] highlight three key factors of 5G that enable data-driven manufacturing: volume, velocity, and variety of data. 5G's bandwidth, speed, and compatibility with diverse data formats make it a formidable technology for real-time decision-making in industrial settings. 5G network slicing enables the creation of logically separated, use-case-specific virtual networks within the same physical network, providing better flexibility and service quality [107]. However, while 5G technologies promise to connect a massive number of devices over long distances, they are limited in that they require infrastructure support and licensed bands [108]. Moreover, IIoT applications typically require relatively small throughput per node, as capacity is not the primary consideration. Rather, the primary focus concerns the capacity to connect an extensive array of devices to the Internet at a low cost, while accounting for limited hardware capabilities and energy resources [109]. This highlights the importance of prioritizing features such as latency, energy efficiency, cost-effectiveness, reliability, and security within the realm of PnP systems.

2.3 DISCUSSION AND GAP ASSESSMENT

In this section, we present a discussion on the challenges associated with developing and enabling PnP systems. Subsequently, we suggest future research directions based on the identified gaps from the discussion.

2.3.1 CHALLENGES AND POTENTIAL SOLUTIONS

Despite recent advances, challenges persist in realizing seamless integration of IIoT solutions in industry. The development and implementation of enabling technologies from

the covered literature bring forth several challenges that directly impact the interoperability, scalability, and mobility of PnP systems. Addressing these challenges is crucial to realizing the full potential of PnP.

The literature shows that extensive research has been conducted on the design of PnP systems, with numerous standards developed to attain comprehensive interoperability across systems. However, challenges arise when applying these standards to IIoT peripherals with limited computing capacity. This limitation stems from the insufficient computing capability to process these protocols and handle data representation. For example, while OPC UA incorporates both the data modeling and data transport aspects of IIoT communication, legacy devices as well as IIoT peripherals with limited computing capacity are unable to process the device capabilities represented within the OPC UA information model. In cases such as these, dedicated edge devices can be utilized to take on the task of processing device information to enable various ACMs. Moreover, lightweight communication protocols such as MQTT can further reduce the strain on the processing device. This can be incorporated into existing OPC UA infrastructures either through some kind of translation mechanism, or more simply by using the PubSub extension of the OPC UA protocol.

For cloud computing infrastructures, the literature suggests that despite its advantages, challenges persist in enabling cloud-centric IIoT applications for PnP systems, specifically concerning real-time task prioritization, resource discovery, and standardization of design approaches. Edge computing technologies can help to mitigate these issues by utilizing the local processing capabilities for critical IIoT applications that have real-time processing requirements. For more dynamic environments, edge computing

can be combined with task offloading techniques to correctly prioritize IIoT tasks according to the specifications of the manufacturing scenario.

The literature also mentions disadvantages regarding edge computing architectures; researchers highlight issues related to stateful programmable data-plane abstraction, transportation of large-scale data, resource allocation, scalability, integration, energy consumption tradeoff, and data monitoring. Most of these issues can be mitigated through the combination of edge computing and cloud computing infrastructures and simultaneously utilizing task offloading services to allocate resources accordingly. However, task offloading techniques are typically use-case specific and require intensive manual configuration to integrate into manufacturing scenarios.

Lastly, investigations on ACMs in PnP scenarios have typically been focused on device discovery capabilities. Recently there has been research into providing more functionality for the various ACMs, such as allowing for discovery and virtual representation of heterogeneous devices that operate using differing frameworks. While these ACMs enhance the scalability of the PnP system across the equipment layer, there still lies the issue of exposing the addition of new devices to the system to applications in higher layers of the IIoT architecture that analyze this data, such as business applications in the cloud.

2.3.2 FUTURE DIRECTIONS

Based on the identified gaps in the literature from the previous section, a few research opportunities are presented for future development of PnP systems:

Table 2.2: Future Directions of PnP

Application of ACMs to Task-offloading	As more devices are added to IIoT infrastructures, the scalability requirements of the system become more demanding. However, while task offloading techniques can be used to address the increased scalability requirement, task offloading techniques are typically use-case specific and require intensive manual configuration to integrate into manufacturing scenarios. Therefore, PnP systems could potentially benefit from research regarding the combination of ACMs with task offloading techniques to address this issue. Furthermore, for wide scale integration of task offloading techniques to be incorporated into current industrial environments, more research must be conducted on the development of generalized task offloading frameworks that can be applied to various domains within the IIoT.
ACMs in Hybrid Cloud-Edge Infrastructures	ACMs still face issues regarding extending the mobility they provide to PnP systems to higher levels of the IIoT architecture. Therefore, PnP systems can benefit from investigations into the incorporation of ACMs within hybrid edge-cloud computing infrastructures to facilitate the interaction of these ACMs across the entire span of the IIoT architecture.
AI-Enabled ACMs	ACMs require the implementation of autonomous software agents that can communicate, negotiate, and collaborate with each other to achieve system configuration goals and adapt to changing conditions in real-time. Artificial intelligence and machine learning techniques can be incorporated with these ACMS to analyze historical data, predict future system requirements, and optimize system configuration dynamically based on evolving production conditions.

CHAPTER 3

THEORY AND DEVELOPMENT

3.1 INTRODUCTION

As identified from the literature review in the previous chapter, the core characteristics of Plug-and-Produce (PnP) IIoT are interoperability, scalability, and mobility. This thesis focuses on developing a PnP framework that embodies these characteristics for the manufacturing testbed in the Future Factories lab located at the University of South Carolina's McNair Aerospace Research Center. The next section identifies the use-case and requirements for PnP in the Future Factories manufacturing testbed. Following this section, the thesis introduces several elements within the manufacturing testbed, both cyber-physical and software-based, that highlight the necessity of interoperability and seamless information exchange in manufacturing operations. Next, the enabling technologies for the development and implementation of the PnP framework are described. Finally, the thesis discusses the development methodology of the PnP framework for the Future Factories use-case.

3.2 PLUG AND PRODUCE CONNECTIVITY FRAMEWORK

Currently within the manufacturing domain, there exists an extensive number of options for communication and data formatting between manufacturing devices. Centralized solutions for data interoperability tend to be domain-specific and require significant processing resources, making them unsuitable for time-sensitive applications. However, the Future Factories manufacturing testbed incorporates research from several

intersecting domains within Smart Manufacturing, all of which demand high data availability and rich data description. Furthermore, these domains incorporate various software suites and applications that do not provide out-of-the-box interoperability between these heterogeneous devices. Therefore, it is necessary to create optimized hardware and software solutions that are platform-agnostic, vendor-neutral, and is capable of bilateral communication with data repositories to enable the interconnectivity of devices isolated in different vendor silos.

3.2.1 USE-CASE

Adhering to the guidelines described in the previous section, the primary objective of this thesis is to develop a PnP framework that is capable of translating and exposing data between industrial machines and applications to be utilized in the research of future manufacturing applications. For the purpose of enabling interoperability in the IIoT, this thesis investigates the development of custom information models as well as the translation between the OPC UA Framework and IIoT Protocols. To enable scalability for the PnP solution, the middleware takes advantage of several capabilities of the IG software to reduce the manual effort of device configuration and system integration while also enhancing data availability. Furthermore, mobility is provided to the PnP solution by taking advantage of the characteristics of wireless networking, as well as developing the PnP solution on edge devices with a low physical footprint and limited computing resources.

3.2.2 REQUIREMENTS

To create the PnP use-case in the Future Factories manufacturing testbed, the definition of specific requirements is necessary. These requirements were created

according to the primary PnP initiatives identified in Chapter 1 of this thesis, as well as the PnP characteristics defined from the literature. The requirements are detailed in Table 3.1.

Table 3.1: PnP Requirements for the Future Factories

Requirements	Description
Physical Infrastructure	The physical infrastructure of the manufacturing testbed must be designed. The infrastructure must incorporate manufacturing equipment and processing devices of various vendors.
Open-communication	The manufacturing equipment and associated processing devices must possess open-communication capabilities to foster data acquisition and transmission of the manufacturing process data.
Interoperable	The PnP framework must be capable of translating data into a format that is understood by both the sending and receiving systems. Communication between systems should also be bidirectional to enable cohesive interaction between operational equipment and IT applications.
Scalable	The PnP framework must increase the availability of manufacturing data and expose this data to all relevant systems within the testbed. The framework should also be capable of being easily extended to accommodate additional devices.
Mobile	The PnP framework must be developed on a device with a low physical footprint. The middleware should reduce the configuration effort for system integration and should also be easily transferrable between systems.
Horizontal and Vertical Communication	The framework should enhance communication both horizontally and vertically among all hardware and software entities in the IIoT infrastructure to enable seamless interaction between applications and industrial systems.

3.3 FUTURE FACTORIES MANUFACTURING TEST BED



Figure 3.1: Future Factories Testbed

The test environment within the Future Factories lab serves to enable the integration of diverse technologies into smart manufacturing processes and is shown in Figure 3.1. The testbed is a platform to introduce enabling technologies across various research topics within the Smart Manufacturing domain, such as state of the art tools in robotics, IIoT, data analytics, and edge computing. As the testbed is meant to serve as a cornerstone for future manufacturing research, the testbed infrastructure was designed to be reusable and reconfigurable, allowing it to adapt to different use-cases. The testbed features four conveyor belts designed to provide optimal flexibility for manufacturing processes. Currently, the manufacturing process executed by the testbed is an assembly process for a customized 3D-printed model rocket. The testbed is also cross-domain, as it incorporates equipment from a range of manufacturers, including Siemens, IBM, Dell, Yaskawa, and

others. This manufacturing testbed serves to showcase the collaborative capabilities of various software platforms and equipment assets within the testbed.

3.3.1 DIGITAL TWIN

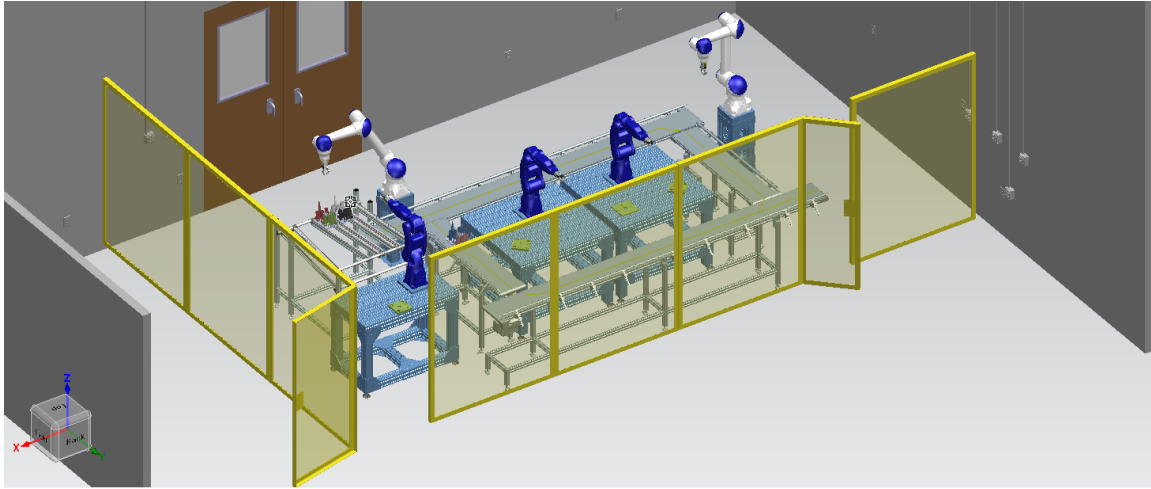


Figure 3.2: DT of the Future Factories Testbed

The manufacturing domain is increasingly incorporating the use of DT technology to digitally represent assets within a manufacturing facility. A DT is essentially a virtual representation of the physical objects, assets, and processes within a system. It is a detailed and dynamic digital model that mirrors the physical facility, providing a means to monitor, analyze, and simulate real-world entities. The model replicates the characteristics, behavior, and attributes of its physical counterpart. The Future Factories manufacturing testbed uses the Siemens Tecnomatix Process Simulate software as the primary DT tool. The software is used to model physical testbed in virtual space, as shown in Figure 3.2.

3.3.2 PLC PROGRAMMING

PLCs serve as the central processing unit for automation processes in the Future Factories testbed. PLCs are controllers designed to monitor and control manufacturing processes, controlling various machines such as robotic devices and conveyors. In the Future Factories testbed, a Siemens S7-1516F CPU is used to control manufacturing

operations, along with several distributed I/O modules for the connection of sensors to the PLC through the proprietary Siemens PROFINET network. The PLC program is designed using a combination of Ladder Logic and Structured Control Language within the Siemens Totally Integrated Automation (TIA) Portal engineering software.

3.3.3 VISUAL INSPECTION

The Future Factories testbed includes a visual inspection station that is located at the beginning of the model rocket assembly process. The station performs an inspection by taking a picture of the incoming rocket tray on a mobile phone to verify that the physical configuration of the assembly work pieces is correct before the assembly process begins. Previously, the Future Factory testbed used the proprietary IBM Maximo Mobile Visual Inspection (MVI) application on an iPhone. However, the proprietary nature of the application does not allow for the application to be modified based on the desired use-case, which makes it difficult to integrate with external open-source applications. As such, Future Factories has created a new platform for students to develop and implement MVI systems by creating a custom visual inspection station.

3.3.4 EDGE COMPUTING

Edge computing is a research domain that brings processing capabilities closer to the data source rather than relying solely on cloud computing technologies. Typically, these processing devices, called edge devices, are mounted near the data sources and directly communicate with operational assets. By situating processing capabilities closer to the data source, edge computing enables real-time processing of the generated manufacturing data. Currently in the Future Factories testbed, a Siemens IPC 227E Industrial PC acting as an edge device is connected to the PLC using the proprietary S7 connection to gain access to

the PLC data. Edge computing is utilized by this edge device to filter the PLC data before transmitting it to the cloud via MQTT. For the development of the PnP framework, edge computing is employed in order to create a local centralized data pipeline for the collection and aggregation of manufacturing data, which is then distributed to the interested client applications on the network.

3.3.5 CLOUD COMPUTING

Cloud computing refers to the capability of executing computationally intensive tasks over the internet, or in the cloud. Typically, this functionality involves the deployment of infrastructure such as servers that are hosted on the internet. Cloud computing serves various purposes, including data storage, analytics, and handling complex intelligence tasks. It proves particularly valuable for consolidating data from diverse physical locations into a centralized server. The Future Factories manufacturing testbed integrates two cloud computing platforms, which are the Siemens Insights Hub and IBM Maximo Application Suite platforms.

Cloud computing is primarily employed in the testbed for establishing the connection of data from the various data sources to the different cloud platforms. Cloud computing assumes the central role as the platform for advancing reasoning capabilities, particularly in generating contextualized information from the edge. Currently, the Siemens Insights Hub platform pushes the manufacturing data from industrial machines (robots, conveyor VFDs, etc.) involved in the rocket assembly process to the Insights Hub cloud, where the manufacturing data is analyzed for trends and is visualized through dashboarding. IBM Maximo Application Suite, on the other hand, collects information

from the MVI application in the testbed and sends it to the Maximo Application Suite cloud.

3.4 OPC UA AS AN INTEROPERABILITY SOLUTION

The foundation of OPC UA rests on 2 major pillars, which are data modeling and the transportation mechanism. Data modeling refers to how to describe the data hierarchy and relationship of virtual objects requested from the server and occurs within the Framework Layer of the IIoT Communications Stack. This concept establishes a shared framework for constructing a more intricate and structured information model for the process data handled by industrial devices [62]. This framework simplifies the hierarchical representation of complex data types, and the resulting information model is stored within the address space of an OPC UA server, which can be presented in an organized manner to any OPC UA client.

The transportation mechanism, on the other hand, describes the mechanism by which data is transported between OPC UA systems, and operates within the transport layer of the IIoT Communications Stack Model. The transportation mechanism facilitates the transmission of machine data, such as control variables and parameters, in a machine-readable format while also preserving the semantic annotation of information. These pillars collectively define OPC UA as a robust and versatile framework for facilitating information sharing and data communication, ensuring compatibility across platforms, and supporting complex information models within a service-oriented architecture.

3.4.1 OPC UA FRAMEWORK

The OPC UA Framework is classified into 4 specification types – Core, Access Type, Utility, and Companion – that are divided further into various parts, each serving distinct purposes [73]. These specification types are described in Table 3.2.

Table 3.2: OPC UA Framework Specification Types

Core Specification Parts	Form the foundational components of OPC UA technology. We focus on the OPC UA Core Specification Parts that relate to the enablement of interoperability and connectivity in the framework, notably Parts 3-6.
Access Type Specification	Standardized OPC-specific information models tailored for providing classic OPC information, such as Data Access, Alarms, etc.
Utility Specification Parts	Encompass additional tools to enhance the functionality of OPC UA, such as server discovery.
Companion Specifications	Additional information models defined for specific use-cases or industries that are developed via collaborative efforts from organizations promoting framework and protocol standardization.

Together, these specifications create a comprehensive framework for implementing OPC UA, catering to various aspects of information modeling, data access types, utilities, and collaborative standardization efforts. The collaborative approach for developing modular and use-case-specific information models ensures that OPC UA extends its interoperability and applicability by accommodating diverse industry standards and requirements.

OPC UA is designed to be cross-platform and internet-ready, boasting firewall-friendly attributes by utilizing established protocols like HTTPS. The framework incorporates a complex information model and follows a service-oriented architecture. The OPC UA server exposes micro-services in the form of methods, allowing clients to request

information seamlessly. For instance, methods like ReadTag(), WriteTag(), and FindServer() enable access to read data services, write data services, and configuration server services, respectively. Another key characteristic is simplified IT integration, addressing the flow of data between field devices on the shop floor and higher-level applications such as Enterprise Resource Planning (ERP) software.

3.4.1.1 OPC UA BASE INFORMATION MODEL

The information model concept is to create a comprehensive hierarchical structure enabling the digital description of objects and processes within the manufacturing environment. Within this hierarchy, the basic unit of information is the *node*, serving as the fundamental element for constructing various types of information. Nodes play a pivotal role in the creation of variables, methods, and objects, as everything in the OPC UA address space is considered a node.

An essential component within the OPC UA Framework is the OPC UA Base Information Model, which consists of *base node classes*. Base node classes specify universal metadata characteristics that all nodes possess, including the *node id* which serves as the numeric identifier for the node, *display name* which acts as a descriptive alias for the node, *node class* which classifies the type of information the node represents, *references* which define the relationship between different nodes, and *browse name* which is used to search for the hostname in the OPC UA address space. The base node classes encompass various types such as *variable*, *variable type*, *object*, *object type*, *view*, *data type*, *reference type*, and *method*.

Furthermore, Nodes can be enriched with additional information in the form of properties, allowing for a more detailed and comprehensive description of each node's

characteristics. Nodes are interconnected through References that are classified by Reference Type, establishing relationships and connections between different nodes within the OPC UA address space. This structured approach to defining Nodes and their attributes, along with the interconnections through typified References, forms the foundation for creating a well-organized and interconnected OPC UA address space, facilitating effective communication and data exchange within the system.

However, there is a notable weakness associated with the standard OPC UA Client-Server interaction pattern; while an OPC UA client can dynamically explore and map out the information model within the OPC UA Server's address space, this can only occur after establishing a connection to the server. To ensure true interoperability between heterogeneous systems, a system must possess the capability to anticipate the information structure it will encounter even before establishing a connection. This is where the development of standardized companion specifications plays a role in enabling interoperability.

3.4.1.2 OPC UA COMPANION SPECIFICATIONS

OPC UA *companion specifications* play a crucial role in achieving interoperability in industrial automation and communication systems. Companion specifications define standardized information models and data structures for specific industries or use-cases, ensuring that different devices and systems can seamlessly exchange data and understand each other. OPC UA companion specifications are developed for 2 primary reasons [74]:

1. To publish information models for a specific use-case e.g., in a particular industry, or for a particular set of devices.
2. To specify how to use OPC UA in specific environments e.g., defining a cloud-based library of various OPC UA Information Models and Namespaces [75].

By adhering to OPC UA companion specifications, vendors can implement standardized information models in their devices and systems. This facilitates interoperability between products from different manufacturers, allowing them to communicate and exchange data seamlessly. With OPC UA companion specifications, integration efforts are reduced because devices and systems follow a standardized set of rules for data representation and communication. This minimizes the need for custom interfaces, proprietary protocols, and extensive integration efforts when connecting new components to existing systems. OPC UA companion specifications also contribute to the PnP capability in that they allow devices to be easily integrated into existing systems. This simplifies the deployment of new equipment, reduces configuration complexities, and supports dynamic reconfiguration of systems.

3.4.1.3 DESIGNING A CUSTOM INFORMATION MODEL

Building upon definitions from developed industry-standard companion specifications, OPC UA also provides interoperability to custom information models. Custom information models built from the framework of existing standardized companion specifications, such as the Device Integration (DI) information model [76], serve to enable heterogeneous devices and applications to consistently comprehend information from each other. Through building *nodes* based on the *node classes* from the companion specification

- which are in-of-themselves built from the *base node classes* from the core specification
- the information model can define relationships between nodes that are understood by both systems. This anticipatory strategy is crucial for fostering seamless communication and understanding among diverse components within a system.

Designing a custom OPC UA information model for a complex system involves a step-by-step process with the ultimate aim of generating a NodeSet2.xml file [77]. If companion specifications are used, then the information model will also generate Types.bsd and NodeIds.csv files to describe the unique data types and assign the Node Ids for each node in the address space, respectively. There are generally three methods to achieve this, as shown in Figure 3.3.

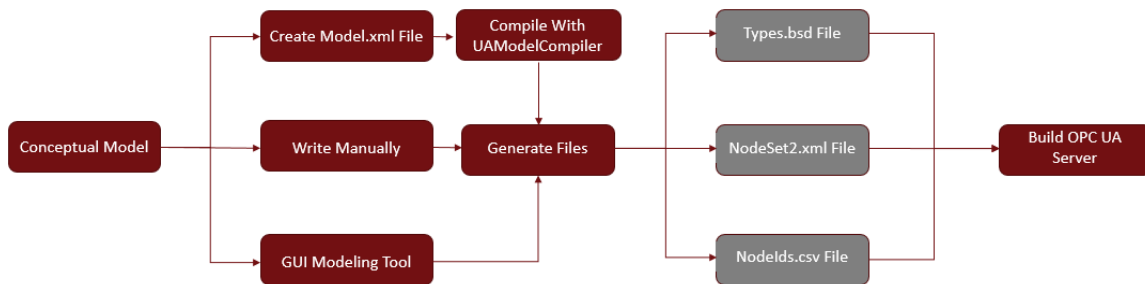


Figure 3.3: Design Process of Custom Information Model

Firstly, one can opt for the manual approach, which involves manually typing the NodeSet2.xml, Types.bsd, and NodeIds.csv files. However, this method requires significant experience in OPC UA and is also the most difficult method for debugging errors. Alternatively, a graphical design tool such as UAModeler can be employed [78]. With this approach, users graphically design the information model, and the tool automatically generates the necessary NodeSet2.xml file. However, these tools are generally only freely available when building models with an extremely limited amount of nodes. As custom information models often use a combination of several companion

specifications simultaneously, GUI modelers can limit how manufacturers can define the information hierarchy in the model. Lastly, there's the option of manually creating a Model.xml file to define the structure of the information model. To execute this method, the UA Model Compiler tool is necessary to compile the Model.xml into a NodeSet2.xml file, accompanied by the required .NET code.

Each of these methods offers a unique approach to designing custom OPC UA information models, catering to different levels of expertise and preferences in the development process. This research work opted to use the 2nd and 3rd method for designing the information models, as we found the generated NodeSet2.xml file from these methods to be the most stable when incorporating the information model with other open-source libraries. More specific detail on the design of the information models for the Future Factories manufacturing testbed is provided in Chapter 4.

3.4.2 OPC UA TRANSPORT

Interoperability on the transport layer is achieved through the standardization of different transport protocols, such as TCP/IP and HTTP(S), in combination with different encoding formats e.g., JSON, BIN, and XML [10]. Users are then able to interact with the data through standardized interaction services, such as reading and writing the attributes of nodes in the OPC UA address space. However, since the IIoT still lacks a generic protocol translation tool [9], there is still a lack of horizontal communication between devices segregated into vertical silos of proprietary systems, as the current state of the IIoT infrastructure lacks suitable methods to provide interconnectivity at both the framework and transport layers in multi-protocol settings [79]. To establish semantic uniformity between these segregated silos, there are 2 conditions that must first be met:

1. There must be a mapping between the OPC UA address space and the IIoT protocol message structure.
2. There must be a mapping between the OPC UA services and IIoT protocol services.

These 2 elements are investigated for the development of the PnP prototype. The MQTT Sparkplug protocol was chosen as the IIoT protocol for development of the PnP prototype due to its capability to automatically define the hierarchical topic structure with respect to the nodes of the OPC UA information model.

3.4.2.1 MAPPING BETWEEN THE OPC UA ADDRESS SPACE AND MQTT PROTOCOL MESSAGING

When translating information and data between OPC UA and MQTT, it's essential to understand that the OPC UA Framework and MQTT protocol serve different purposes and have distinct infrastructures. OPC UA is designed for complex object-oriented industrial communication and represents relationships between nodes through references, whereas MQTT is a lightweight messaging protocol most commonly used for simplified device-to-device communication. Considering that the OPC UA Framework has comparatively much more advanced data modeling capabilities, there are issues when translating between OPC UA and MQTT. What is the best way to translate these metadata-rich representations to the MQTT infrastructure? To represent the OPC UA information model when translating to MQTT, we must consider a mapping strategy that fits the hierarchical, publish/subscribe nature of MQTT.

While the OPC UA address space is composed of nodes and references that define the relationships between these nodes, MQTT represents the nodal hierarchy as a topic

hierarchy, where each node in the OPC UA address space is mapped to a unique topic in MQTT. A simple method was utilized to represent OPC UA nodes in the MQTT infrastructure. This method is to encapsulate the node information – i.e., attributes and properties – in the payload structure of the MQTT messages. With this method, the MQTT payload structure is designed to represent relevant node attributes as metrics of the MQTT Topic for that Node i.e., the *value attribute* of the OPC UA Node is represented as the *value metric* of the MQTT message payload.

3.4.2.2 MAPPING BETWEEN OPC UA AND MQTT PROTOCOL SERVICES

The mapping between the abstract descriptions of OPC UA services and the communication stack derived from these services are defined by the OPC UA Core Specifications [80]. OPC UA services are organized into *service sets* that define a set of related services e.g., the *discovery* service set defines services that allow clients to discover endpoints implemented by a server. One service set of interest for the PnP prototype is the *attribute* service set, which defines services that allow OPC UA clients to read and write the attributes of nodes. As the value of a variable node in the OPC UA server is defined as a *node attribute*, this service can be used by an OPC UA client to read and write the values of variables in the OPC UA server. When this occurs, the changing of the variable value attribute must be reflected in the changing of the value metric for the associated MQTT topic.

The *node management* service set defines services that allow an OPC UA client to add, modify, and delete nodes in the OPC UA server address space, meaning that clients can dynamically change the structure of the information model upon introducing new machines to the industrial environment. However, dynamic changes to the information

model in the OPC UA server address space must also be reflected by changes to the MQTT topic structure, such as generating a new MQTT topic when a new node is added to the OPC UA server.

3.5 IGNITION GATEWAY AS A SCALABILITY SOLUTION

Ignition is an integrated software platform for Supervisory Control and Data Acquisition (SCADA) systems developed by Inductive Automation. The platform features a cross-platform web-based deployment through the cooperation of IGs, the Ignition Designer application, and several runtime clients. A key characteristic of Ignition is the utilization of independently developed modules that provide functionality to individual components within the Ignition platform. The Ignition platform includes several capabilities that help to enable scalability for IIoT solutions. These capabilities include the transformation of devices of various operating systems (Windows, Linux, MacOS) into IGs to enable communication among them, the utilization of select PLC drivers to access information from PLC address spaces, the bridging of distributed data sources into a centralized OPC UA architecture, and the translation between various communication protocols e.g., HTTP/HTTPS, TCP/IP, WebSockets, MQTT, etc. In combination with the interoperability enabling modalities described in the previous sections, the IG extends the interoperability of the PnP solution in a scalable manner.

3.5.1 IGNITION GATEWAYS

The central software service governing all operations in the Ignition platform is the IG [117]. This singular application functions as a web server that is accessible from any web browser. Its functionalities encompass tasks such as establishing connections with data sources and PLCs, executing modules, and communicating with clients. The IG is accessed

through a web browser via the *gateway web interface*. The web browser, running on any machine, must have network access to the IG host. From this web browser, the IG can be configured to manage connections to new data sources/sinks, such as a database connection to an SQL server, connections to 3rd party OPC servers, and device connections to production machines given that Ignition includes the drivers to communicate with the device architecture.

The IG also allows users to establish a *gateway network*, enabling the connection of two or more IGs. The gateway network allows users to connect multiple IGs together over a wide area network, facilitating various distributed interactions between IGs [118]. The gateway network uses a dedicated HTTP data channel capable of handling multiple streams of message data. By configuring incoming and outgoing connections between endpoints of different IGs, the gateway network allows for seamless data sharing between the IGs.

3.5.2 IGNITION TAGS

The Ignition architecture defines the data points from various data sources as *Ignition Tags*. These can be defined depending on the protocol of the data source e.g., OPC Tags, SQL Tags, etc. At the highest level of tag configuration is the *Tag Provider* [119]. Tag Providers are a collection of Ignition Tags that are organized according to the data source. Tag Providers are categorized into 2 types, depending on which gateway is storing the data and which gateway is accessing the data:

1. **Standard Tag Providers:** Collections of Ignition Tags from various data sources handled locally in the IG.

2. **Remote Tag Providers:** Establishes a connection to a remote IG and retrieves tag information. A link is created between the local IG to a Standard Tag Provider on a remote IG through a gateway network connection.

These Tag Providers essentially act as clients that are capable of subscribing to various types of data sources, translating the heterogeneous data into Ignition Tags. In the proposed PnP framework, these Ignition Tags act as an intermediary format, as the newly formed Ignition Tags can then be translated into the OPC UA Framework in the local Ignition OPC UA Server. While Tag Providers are configured within the IG webserver, the actual Ignition Tags themselves are created and managed within the Ignition Designer application.

Ignition Designer is an application for interacting with the data published to the IG. The software enables a wide variety of applications, including but not limited to SCADA data visualization and analysis tools, scripting, report generation, SQL querying, etc. Ignition Designer includes the essential Tag Browser tool, which allows for the creation of Ignition Tags from various data sources, such as an OPC UA server, SQL queries, variables derived from scripting, and data received through a direct device connection to the IG using the aforementioned device drivers.

3.5.3 IGNITION OPC UA SERVER

The IG can access data from industrial machines by configuring various device connections. When configuring a device connection, Ignition includes several driver modules that allow connections to specific types of devices from the Ignition OPC UA server. Some of these drivers include Allen-Bradley over Ethernet, Modbus TCP and RTU,

Siemens S7, BACnet, etc. Using these drivers, Ignition connects to these devices using the associated protocol for the device. These drivers enable the writing of machine data into Ignition Tag Providers without the need for intensive manual configuration.

However, it is important to note that a few of these drivers do not support tag browsing of the device, such as Siemens and Modbus, meaning that manual configuration of the OPC path for the Ignition Tag is required [120]. In cases such as these, data acquisition can be handled by configuring a 3rd party OPC UA server to store the associated device's process data. IGs can create connections to and access data from external OPC UA servers, acting as both an OPC UA server and an OPC UA client. These OPC UA connections can be classified as separate Tag Providers and bridged to the Ignition OPC UA server. Moreover, the Tag Provider containing the data from external OPC UA servers preserves the structure of the information model for the device, allowing for the aggregation of information models from heterogeneous systems onto the centralized OPC UA server on the IG.

3.5.4 CIRRUS LINK MQTT MODULES FOR IGNITION GATEWAYS

The IG includes several additional modules to enable further communication capabilities to an industrial environment. Among these additional modules are the MQTT Distributor, MQTT Transmission, and MQTT Engine modules developed by Cirrus Link Solutions. These modules for Ignition are designed specifically for the integration of MQTT data in building IIoT solutions [121]. Using these MQTT modules, Ignition allows the user to build and deploy a flexible and scalable IIoT architecture. Although there are many possible variations in which the IIoT architecture can be designed, the core of the solution involves 3 basic steps:

1. Access data from the edge of the network.
2. Publish data from the edge of the network to the MQTT server.
3. Publish and Subscribe to data from the MQTT server from external applications.

This process occurs by utilizing the distinct features of each of the aforementioned Cirrus Link MQTT modules.

3.5.4.1 MQTT DISTRIBUTOR MODULE

The MQTT Distributor module is an MQTT Server, and essentially acts as a local MQTT broker on the IG. MQTT brokers play a key role in facilitating communication and message exchange among devices and applications. The MQTT Distributor module acts as an intermediary central hub that receives, stores, and forwards messages between MQTT clients. In contrast to cloud-based MQTT brokers that operate over the internet, a local MQTT broker is confined to a specific local network. This makes the MQTT Distributor module suitable for use-cases in which devices must communicate over a local network without being exposed to the outside world, as is often the case for industrial environments. Moreover, the MQTT Distributor module includes various configuration settings allowing the users to customize their IIoT solution based on their needs. The MQTT server uses TCP for connections by default but can be configured to enable Websocket connections to enable communication with higher-level applications that communicate over Websockets. TLS can also be enabled to ensure that information is protected from unauthorized access or interception during message transit, as well as to enable authentication of the communicating parties.

3.5.4.2 MQTT TRANSMISSION MODULE

The MQTT Transmission module enables the translation and transmission of MQTT Sparkplug topics and Ignition Tags. The module essentially acts as a bridge between Ignition Tag and MQTT Sparkplug topics. Transmitters are configured to pull data from a specified Ignition Tag Provider, meaning that MQTT messages can be generated from various data sources, including OPC tags, SQL tags, etc. A transmitter acts as a listener and monitors the values of Ignition Tags, and upon a value change, generates MQTT Sparkplug messages that are published to a Sparkplug-enabled MQTT Server [122]. This is typically the MQTT Server from the MQTT Distributor module, but any external broker compatible with the Sparkplug MQTT format can be used. Optionally, transmitters can be configured to use a custom namespace instead of the default SparkplugB namespace for cases in which users prefer using a custom message payload structure. Transmitters configured in the MQTT Transmission module also listen to commands from MQTT Sparkplug messages, allowing for bidirectional communication between publishers and subscribers.

3.5.4.3 MQTT ENGINE MODULE

The MQTT Engine module allows users to enhance the MQTT infrastructure of their IIoT solutions by providing a path to deliver data to both operational equipment and higher-level applications [123]. The module uses the IG to push the MQTT data to the edge of the SCADA network. This essentially creates a single data pipeline for all data to be utilized within the Ignition platform, increasing data throughput while also reducing the effort of data acquisition across the industrial environment.

3.6 PLUG AND PRODUCE FRAMEWORK FOR FUTURE FACTORIES

This section will provide more detail regarding the PnP use-case. First, the development methodologies for the PnP framework are described. This is followed by the description of the architecture for the PnP framework.

3.6.1 DEVELOPMENT METHODOLOGY

The development methodology for the proposed PnP framework of the Future Factories manufacturing testbed is split into 2 parts, for the development of the AGs and the IG, respectively. Together, these methodologies describe a platform for enabling interoperable and scalable information exchange across the manufacturing ecosystem.

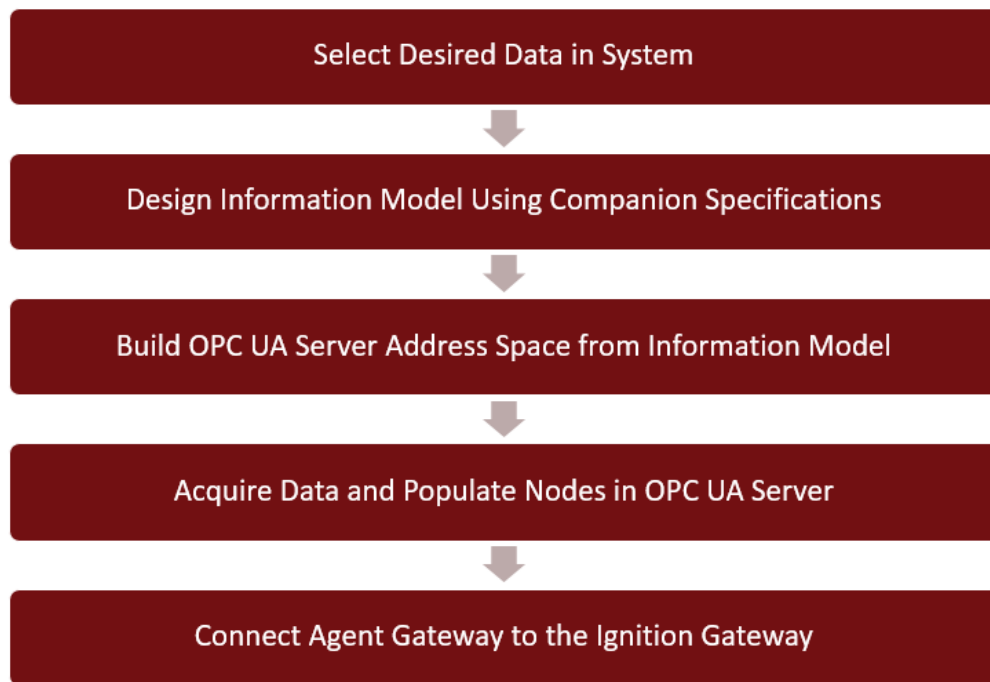


Figure 3.4: Development Methodology for Agent Gateways

Figure 3.4 shows the development methodology for the creation of AGs that utilize OPC UA to enable interoperability between heterogeneous systems. The first step of building the AG is for the user to select what data is required to build a rich object-oriented

data model of the industrial environment. The second step is to create an information model that contains all of the necessary objects, variables, properties, and references to describe the industrial assets in the system. Depending on the manufacturing use-case, different companion specifications can be selected to more accurately describe the properties and capabilities of the industrial assets. The next step is to build an OPC UA server application that imports the developed information model to create nodes with the address space. After building the OPC UA server address space, the server is deployed onto the AG. The next step is to acquire the relevant data and populate the nodes of the OPC UA server address space. Depending on the architecture of and protocol used by the data source, this step can be accomplished in a variety of ways. For the Future Factories testbed, this was accomplished by developing an MQTT client application to subscribe to the MQTT manufacturing data, which is then regularly published to each node in the OPC UA address space. The final step is to connect the AG and IG devices to the same network.

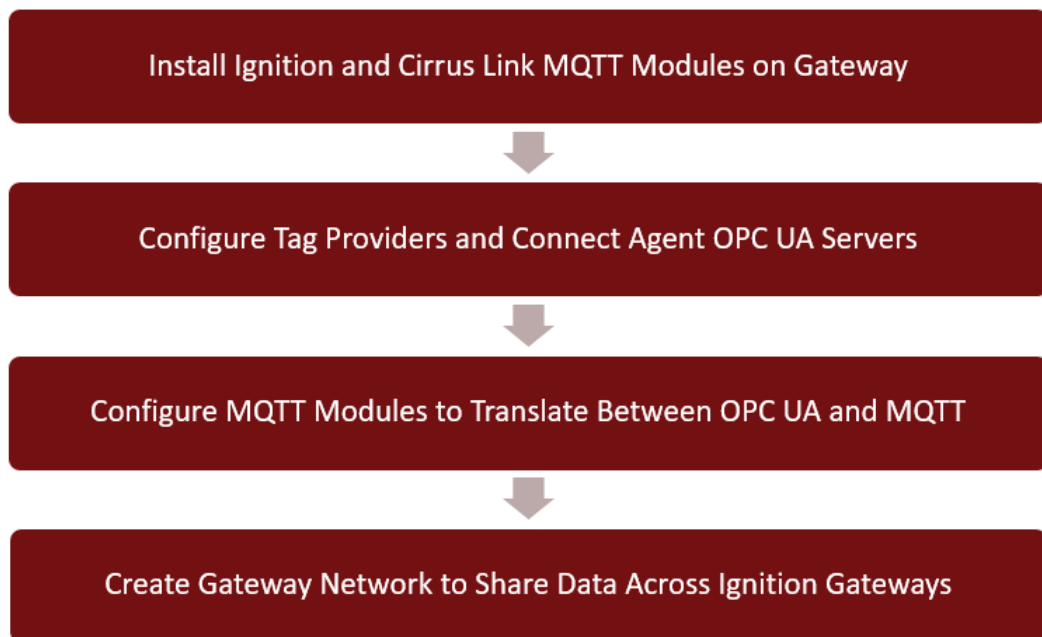


Figure 3.5: Development Methodology for IG

Figure 3.5 shows the development methodology of the central IG for the aggregation and consumption of the manufacturing data transmitted from the AGs. The first step of developing the IG is to install the Ignition software, as well as the Cirrus Link MQTT Distributor, Transmission, and Engine modules onto the PnP middleware serving as the central gateway. After this is done, a Tag Provider is created for each AG, and the OPC UA server from each AG is added as a data source within the IG webserver, allowing Tag Providers to access data from the AGs. The Tag Provider is configured to be exposed to the local Ignition OPC UA server on the PnP middleware.

Once the Tag Provider is receiving the data, the MQTT modules for the IG are configured to set up the MQTT infrastructure. The MQTT Distributor module creates a local MQTT broker on the IG. The MQTT Engine module creates a centralized data pipeline for Ignition applications to consume the MQTT data. The MQTT Transmission module is used to create a transmitter for each Tag Provider. The transmitter serves to translate the Ignition tags from each Tag Provider into MQTT topics that can be subscribed to from any MQTT client on the network. The transmitter also allows for MQTT messages to be published to the MQTT topic, which is then subsequently translated back into an Ignition tag, allowing for bidirectional translation and communication between the internal Ignition OPC UA server and external MQTT clients. Lastly, a gateway network is configured to enable data access to other IGs, facilitating data availability between different facilities and contributing to the Factory-to-Factory scalability concept.

3.6.2 ARCHITECTURE

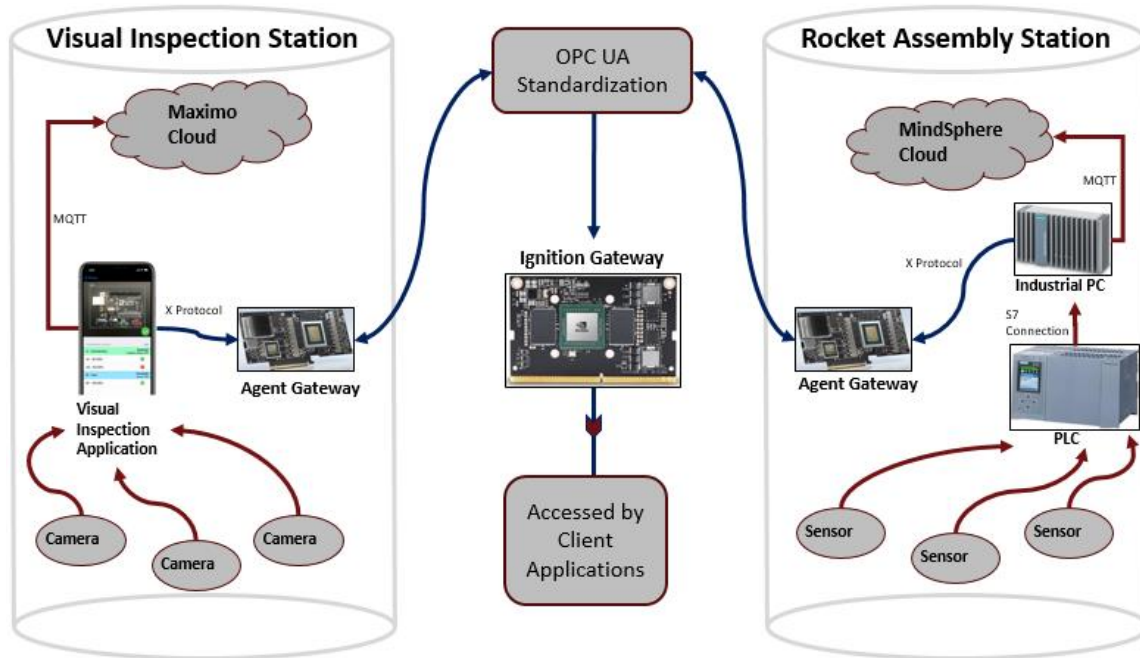


Figure 3.6: PnP Architecture for Future Factories

The architecture of the developed PnP solution for the Future Factories testbed is shown in Figure 3.6. For this use-case, the architecture serves to aggregate and distribute the data for the Rocket Assembly station and the Mobile Visual Inspection (MVI) station in the manufacturing testbed. The “X” in “X Protocol” serves to showcase that the AG can receive the process data from the related application via several different protocols, such as MQTT, OPC UA, HTTP/HTTPS, API calls, etc.

The Rocket Assembly station is controlled primarily by the S7-1516F PLC Program in TIA Portal and serves to autonomously assemble and disassemble the parts of the model rocket in a continuous loop. The assembly process features the use of YASKAWA GP8 and HC10 collaborative robots for fine manipulation of the product work pieces. The robot GPIO, along with other external sensors such as potentiometers and load sensors, are assigned to PLC tags within the PLC address space. The testbed also features

4 conveyor belts to move the work pieces between the robots and are each controlled by a SIMATIC G120C VFD. Parameters from the VFD, such as drive temperature and conveyor speed, are also assigned to the PLC address space. The PLC data is transmitted to the IPC227E Industrial Edge (IE) Databus via the proprietary S7 connection. The Industrial PC then publishes the data from the IE Databus to an MQTT broker under the “FF” topic. On the AG, an MQTT client subscribes to the “FF” topic on the MQTT broker and populates the address space nodes of a local OPC UA server. The AG is then connected to the centralized IG.

The MVI station sits next to the first robot in the Rocket Assembly station and analyzes the configuration of the rocket tray holding the model rocket pieces for detection of defects. This step is performed to ensure the following steps of the rocket assembly operation can be performed correctly. The application was developed on a Samsung Galaxy XCover6 Pro android phone, which is responsible for the capturing, storing, and classification of inspection images of the rocket tray based on the developed image classification model. The image classification model was trained using PyTorch and resnet18 as its pre-trained convolutional base, and a custom image binary classifier was added on top of the frozen convolutional base to classify rocket tray images as either “correct” or “incorrect.” The MVI model is then deployed on the android phone using a Docker container. Using MQTT, the Docker container publishes the predicted class of the rocket tray to the “MVI” topic in a local MQTT broker on the AG. On the AG, another MQTT client subscribes to the “MVI” topic on the local MQTT broker. The “correct” and “incorrect” string values are translated into the UA_Boolean data type for the “CorrectConfig” topic – i.e., true or false – and subsequently populates the address space

nodes of the local OPC UA server application. The AG is then connected to the centralized IG.

3.7 INDUSTRIAL INTERNET OF THINGS INTEGRATION IN BROWNFIELD MANUFACTURING SYSTEMS

The proposed PnP framework is designed to be generic to be easily applicable to various manufacturing use-cases. However, depending on the specific needs of these use-cases, the technical requirements of the IIoT architecture may change. Greenfield manufacturing scenarios typically face less issues regarding IIoT integration as they typically are built using the latest IIoT technologies from the ground up [124]. However, brownfield manufacturing scenarios often still face issues regarding deploying IIoT technologies into existing manufacturing facilities to bridge the divide between OT and IT systems.

Due to the relative youth of the IIoT initiative, many manufacturing facilities are still striving to reach a state of complete automation, and therefore are not yet ready to consider integrating autonomy into their manufacturing solutions. Moreover, the financial and time costs required to transform existing manufacturing systems into full-fledged IIoT solutions often outweigh the benefits associated with achieving full digital synonymy, as implementing these solutions typically involves replacing equipment with smarter alternatives, installation of costly modules to enable open communication for legacy systems, and other requirements that involve increasing production downtime. Reducing these costs is therefore integral for advocating the integration of IIoT technologies in brownfield manufacturing applications. The PnP concept serves to accomplish this goal by reducing the time required to integrate IIoT solutions in manufacturing applications; this

occurs through creating a modifiable and flexible IIoT infrastructure that can react dynamically to changes in the manufacturing environment.

First, it is important to identify problem areas in the manufacturing environment that would benefit from augmenting IIoT technologies. Rather than converting an entire manufacturing facility into an IIoT solution, it is more suitable and economic for brownfield manufacturing scenarios to target specific production lines or already identified issues in the manufacturing facility [124]. Sensors can be mounted onto legacy machines to monitor activities that otherwise would have been performed physically or electrically (activation of valves, actuators, etc.), allowing manufacturers to detect equipment wear and process inefficiencies early and propose corrective solutions.

After adding sensors to monitor the environment and conditions of the problem areas and identifying the data required for the manufacturing solution, the next step is to build a comprehensive information model to describe the characteristics and capabilities of the industrial assets. These assets include product work pieces, industrial equipment, sensors, process parameters, and anything else that serves to describe the industrial environment. The PnP framework uses OPC UA to design the information model, as the OPC UA framework includes several companion specifications that allow manufacturers to precisely describe the industrial process in a digital environment. The information model is used to provide context for the state of the industrial environment to the IIoT system, which can be used to identify manufacturing events upon triggering configured conditions. The information model is then used to build the address space of an OPC UA server.

Once the information models are built for the selected systems, the next step is to begin the process of data acquisition and translation. Because brownfield manufacturing

facilities tend to incorporate IIoT solutions iteratively according to the criticality of the process task, it is common for industrial systems to incorporate communication solutions that are the most compatible with the existing equipment to reduce production downtime. For example, many legacy machines use the Modbus protocol for industrial communication, so manufacturers may be inclined to add sensors that also use the Modbus protocol. However, newer state-of-the-art IIoT sensors often use more common protocols such as MQTT, as MQTT is widely compatible with devices and applications in the modern IIoT ecosystem and allows for much greater customizability and ease-of-use compared to the Modbus protocol. To bridge the communications gap between new, custom, and legacy industrial devices, gateways can be introduced to the manufacturing system to aggregate and translate the different sources of industrial data, freeing manufacturers from being restricted to specific equipment due to interoperability challenges. In multi-protocol settings, as often is the case in brownfield manufacturing scenarios, these gateways must include mechanisms to translate between the various different communications protocols. The proposed PnP framework uses AGs to accomplish this task, using edge computing for acquisition of the industrial data and translation from MQTT into the OPC UA framework. The OPC UA server nodes are then regularly updated from the incoming industrial data.

The final step is to connect the gateways to a centralized data platform to enable communication between the different manufacturing systems in the facility. By doing this, manufacturers are able to develop applications that consume and analyze the different data sources. These applications can be used to identify trends and inefficiencies in the manufacturing facility and facilitate prediction and decision making, such as through the use of machine learning. For example, the state of a digital supply network [17] can be

described within an information model. Upon disruptions to the supply of work pieces required for a specific product, the related industrial processes can then be notified and autonomously modified to account for these changes, such as procuring alternative work pieces to produce the product or scheduling other products to be manufactured until the work piece supply is recouped. In the proposed PnP framework, the IG serves as the centralized data platform for data aggregation and transformation, and also provides the functionality to build SCADA and business applications to analyze the industrial data. The IG can then be optionally connected to MES or ERP systems to provide high-level management of the industrial environment such as through work orders, equipment change requests, financial planning, project management, etc.

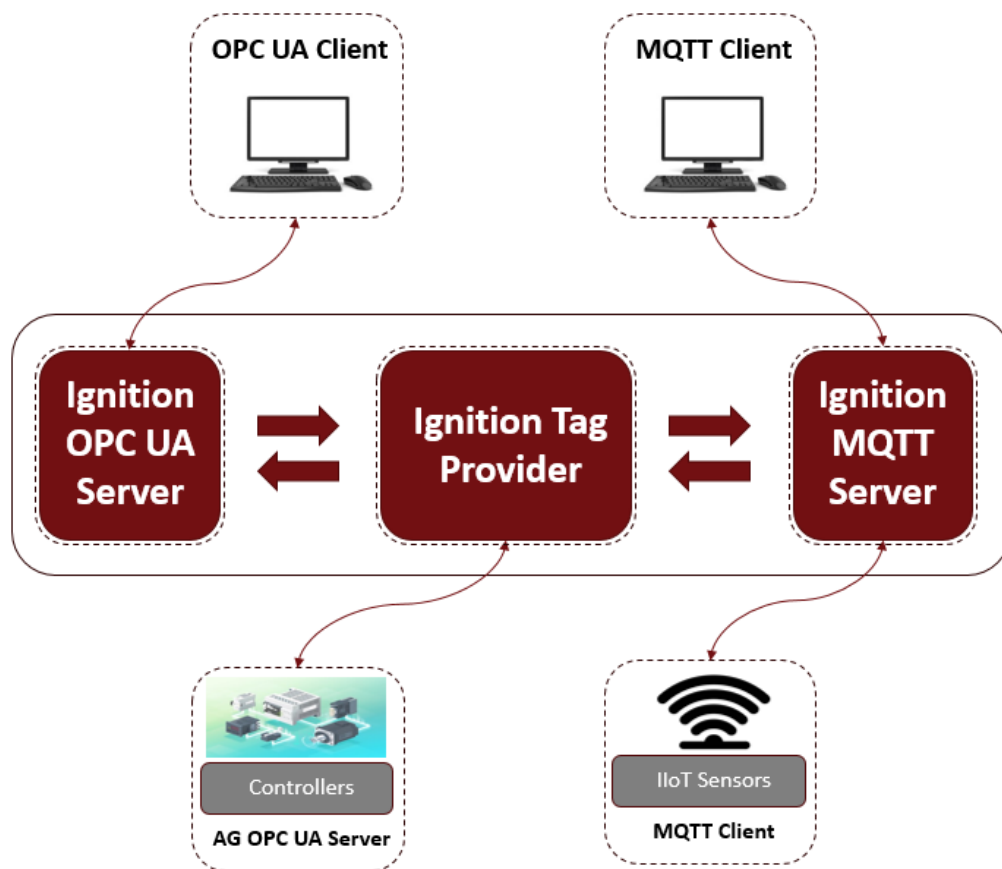


Figure 3.7: Sample IIoT Architecture for Brownfield Manufacturing Systems

Figure 3.7 presents a sample IIoT architecture for the application of the PnP framework for a general brownfield manufacturing scenario. The AG acquires the data from the industrial controllers and other legacy equipment and translates it from the original protocol to the OPC UA framework. Additional IIoT sensors that communicate through MQTT are added to the system to provide more information about the state of the manufacturing process. The IIoT sensors connect to the MQTT server on the IG, whereas the AG OPC UA server connects to a Tag Provider acting as an OPC UA client in the IG, which is then bridged to the IG OPC UA server. The Ignition Tag Provider serves as an intermediate format between the Ignition servers, meaning that changes to the data in either server are reflected in the Ignition Tag Provider, and subsequently in the other server. From here, external applications use the associated client to connect to the Ignition OPC UA or MQTT servers to acquire the data and perform tasks, such as predictive maintenance, process scheduling, business applications, etc.

Altogether, this framework adopts a SoA to allow manufacturers to take advantage of specific services to meet the requirements of the manufacturing system. By incorporating IIoT standardization and translation techniques, the framework enables interoperability across the manufacturing ecosystem. The modularity of the services within the SoA ensures that modifications can be made to the system without disrupting the entire system. This makes it easier to integrate IIoT solutions into existing manufacturing systems, and therefore enhances its scalability and flexibility. The framework facilitates access to data from various sources within the manufacturing environment, which can be aggregated and analyzed to derive actionable insights that drive efficiency, productivity, and intelligent-decision making in manufacturing processes.

CHAPTER 4

IMPLEMENTATION AND DISCUSSION

4.1 INTRODUCTION

This chapter provides a detailed walkthrough of the implementation process for the PnP use-case. Section 2 describes the development of the AGs, while section 3 describes the development of the IG. Section 4 provides a discussion on the aspects of the architecture that contribute to the tenets of PnP described in Chapter 2.

4.2 AGENT GATEWAYS

In this section, the development of the AGs for the Rocket Assembly station and the MVI Station are described.

4.2.1 DESIGNING THE INFORMATION MODELS

In the PnP framework, it is necessary to create information models to digitally describe the properties and capabilities of industrial assets for each respective system. To realize the use-case described previously in Chapter 3, information models are built for the Rocket Assembly station, referred to as the FF information model, and the Mobile Visual Inspection station, referred to as the MVI information model. In this section, the process of creating the information models is described.

4.2.1.1 ROCKET ASSEMBLY STATION

The FF information model is built using the DI and Robotics companion specifications from the OPC Foundation. The Robotics companion specification is dependent on the DI companion specification and is used to define the industrial assets within the Future Factories manufacturing testbed.

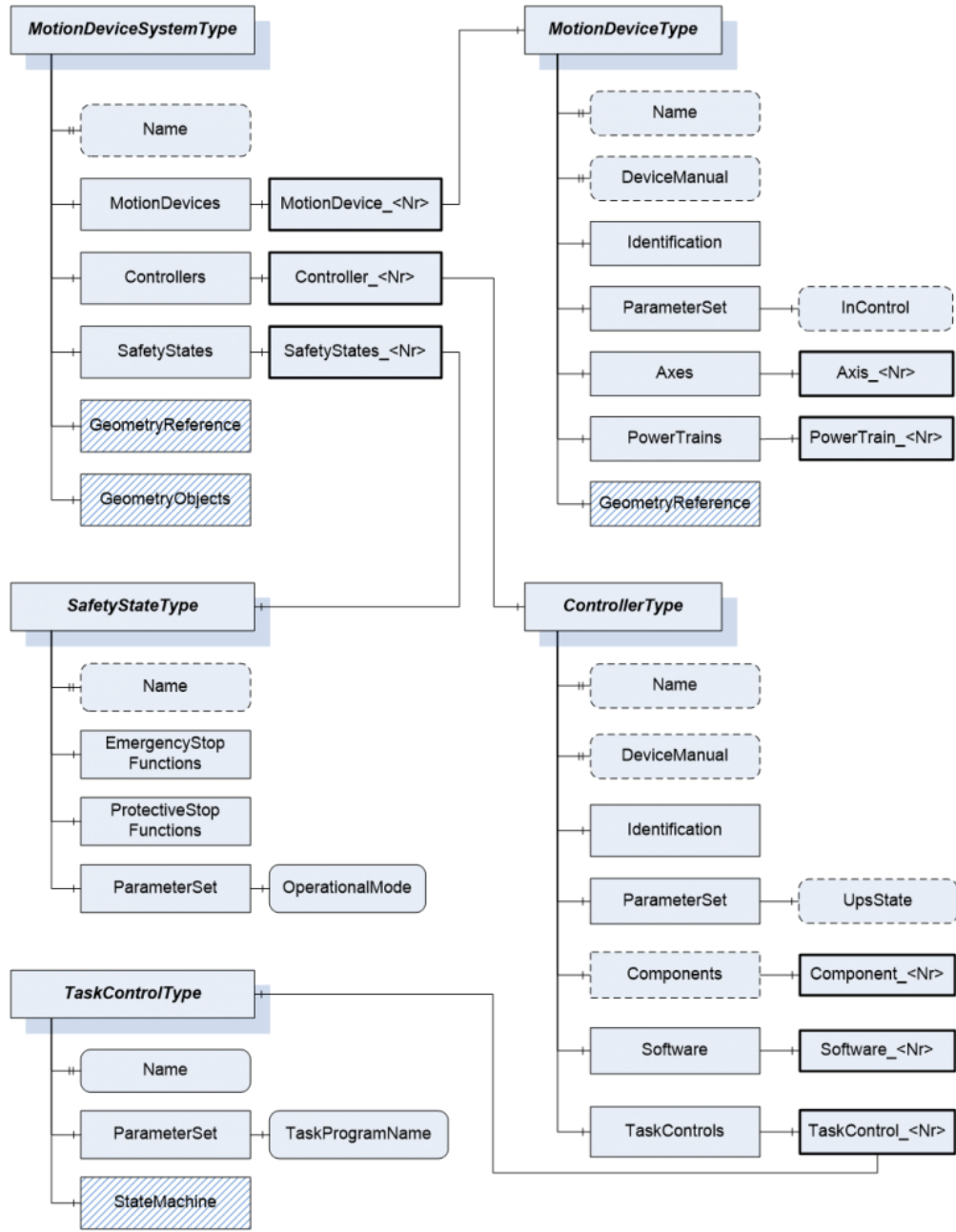


Figure 4.1: Robotics Companion Specification [125]

The Robotics companion specification released by the VDMA Robotics & Automation Association defines several new Object Types, as shown in Figure 4.1. The companion specification includes a basic description of a motion device system by defining the unique MotionDeviceSystemType, MotionDeviceType, SafetyStateType, ControllerType, and TaskControlType Object Types. The MotionDeviceSystemType Object Type is defined to include the Motion Devices, Controllers, and SafetyStates of the motion device system, which are defined according to their associated Object Types. These definitions are used to define the robot and conveyor object instances in the Future Factories manufacturing cell.

The design of the FF information model was created by manually writing a Model.xml file. The purpose of creating the Model.xml file is to define the nodes within the address space of the OPC UA server. The basic procedure for designing the Model.xml is summarized by the following steps:

1. Import or define all of the namespaces required by the custom information model, such as the companion specifications.
2. Define the new namespace object for the custom information model.
3. Define any new Reference Types, Object Types, Variable Types, and Data Types needed to define assets in the custom information model that aren't already defined by the imported companion specifications.
4. Define instances of Objects, Variables, and References to define industrial assets in the information model.

5. Use the UAModelCompiler tool from the OPC Foundation to compile the Model.xml file into the NodeSet2.xml, NodeIds.csv, and Types.bsd files required by the OPC UA server.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <ModelDesign
3   xmlns:uax="http://opcfoundation.org/UA/2008/02/Types.xsd"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xmlns:ua="http://opcfoundation.org/UA/"
6   xmlns:DI="http://opcfoundation.org/UA/DI/"
7   xmlns:ROB="http://opcfoundation.org/UA/Robotics/"
8   xmlns:FF="https://nextusc.com/UA/FF/"
9   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10  TargetNamespace="https://nextusc.com/UA/FF/"
11  TargetXmlNamespace="https://nextusc.com/UA/FF/"
12  TargetVersion="0.9.0"
13  TargetPublicationDate="2018-06-24T00:00:00Z"
14  xmlns="http://opcfoundation.org/UA/ModelDesign.xsd">
15
16  <Namespaces>
17    <Namespace Name="OpcUa" Version="1.03" PublicationDate="2013-12-02T00:00:00Z" Prefix="Opc.Ua" InternalPrefix="Opc.Ua.Server"
18      XmlNamespace="http://opcfoundation.org/UA/2008/02/Types.xsd" XmlPrefix="ua">http://opcfoundation.org/UA/</Namespace>
19
20    <Namespace Name="OpcUaDi" Prefix="Opc.Ua.Di" XmlNamespace="http://opcfoundation.org/UA/DI/Types.xsd" XmlPrefix="DI"
21      FilePath="..\ModelCompiler\Design.v104\OpcUaDiModel">http://opcfoundation.org/UA/DI/</Namespace>
22
23    <Namespace Name="OpcUaRobotics" Prefix="Opc.Ua.Robotics" XmlNamespace="http://opcfoundation.org/UA/Robotics/Types.xsd"
24      XmlPrefix="ROB" FilePath="..\ModelCompiler\Design.v104\OpcUaRoboticsModel">http://opcfoundation.org/UA/Robotics/</Namespace>
25
26    <Namespace Name="FutureFactories" Prefix="Opc.Ua.FF" XmlNamespace="https://nextusc.com/UA/FF/Types.xsd"
27      XmlPrefix="FF">https://nextusc.com/UA/FF/</Namespace>
28  </Namespaces>

```

Figure 4.2: Define Namespaces in Model.xml File

```

30 <!-- Custom Information Model OpcUaFFModel Definition -->
31 <Object SymbolicName="FF:OPCUAConnXtNamespaceMetadata" TypeDefinition="ua:NamespaceMetadataType">
32   <BrowseName>https://nextusc.com/UA/FF/</BrowseName>
33   <Children>
34     <Property SymbolicName="ua:NamespaceUri" DataType="ua:String">
35       <DefaultValue>
36         <uax:String>https://nextusc.com/UA/FF/</uax:String>
37       </DefaultValue>
38     </Property>
39     <Property SymbolicName="ua:NamespaceVersion" DataType="ua:String">
40       <DefaultValue>
41         <uax:String>1.03.0</uax:String>
42       </DefaultValue>
43     </Property>
44     <Property SymbolicName="ua:NamespacePublicationDate" DataType="ua:DateTime">
45       <DefaultValue>
46         <uax:DateTime>2021-03-09</uax:DateTime>
47       </DefaultValue>
48     </Property>
49     <Property SymbolicName="ua:IsNamespaceSubset" DataType="ua:Boolean">
50       <DefaultValue>
51         <uax:Boolean>false</uax:Boolean>
52       </DefaultValue>
53     </Property>
54     <Property SymbolicName="ua:StaticNodeIdTypes" DataType="ua:IdType" ValueRank="Array">
55       <DefaultValue>
56         <uax:ListOfInt32>
57           <uax:Int32>0</uax:Int32>
58         </uax:ListOfInt32>
59       </DefaultValue>
60     </Property>
61     <Property SymbolicName="ua:StaticNumericNodeIdRange" DataType="ua:NumericRange" ValueRank="Array"/>
62     <Property SymbolicName="ua:StaticStringNodeIdPattern" DataType="ua:String">
63       <DefaultValue>
64         <uax:String></uax:String>
65       </DefaultValue>
66     </Property>
67     <Property SymbolicName="ua:DefaultRolePermissions" ModellingRule="Mandatory" />
68     <Property SymbolicName="ua:DefaultUserRolePermissions" ModellingRule="Mandatory" />
69     <Property SymbolicName="ua:DefaultAccessRestrictions" ModellingRule="Mandatory" />
70   </Children>
71   <References>
72     <Reference IsInverse="true">
73       <ReferenceType>ua:HasComponent</ReferenceType>
74       <TargetId>ua:Server_Namespaces</TargetId>
75     </Reference>
76   </References>
77 </Object>

```

Figure 4.3: FF Namespace Object in Model.xml File

The namespaces are defined for each information model referenced by the new custom information model, where “ua” represents the base information model namespace from the OPC UA Core Specifications, “DI” and “ROB” represent the DI and Robotics namespaces from the associated companion specifications, and “FF” represents the Rocket Assembly station namespace. The definition of the namespaces are shown in Figure 4.2. The definitions for the namespace objects of the companion specifications are imported from their respective information models, as shown by the “FilePath” variable in Figure 4.2. After the namespace of the new information model has been declared, the properties of the namespace object for the custom information model are defined, as shown in Figure 4.3.

```

92 <!-- ### Object Types ### -->
93
94 <ObjectType SymbolicName="FF:HC10" BaseType="ROB:MotionDeviceType">
95   <BrowseName>&lt;HC10&gt;</BrowseName>
96   <Children>
97     <Object SymbolicName="DI:ParameterSet" TypeDefinition="ua:BaseObjectType" ModellingRule="Mandatory">
98       <Children>
99         <Variable SymbolicName="FF:JointAngleS" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
100        <Variable SymbolicName="FF:JointAngleL" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
101        <Variable SymbolicName="FF:JointAngleR" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
102        <Variable SymbolicName="FF:JointAngleU" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
103        <Variable SymbolicName="FF:JointAngleB" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
104        <Variable SymbolicName="FF:JointAngleT" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
105      </Children>
106    </Object>
107  </Children>
108 </ObjectType>
109
110 <ObjectType SymbolicName="FF:GP8" BaseType="ROB:MotionDeviceType">
111   <BrowseName>&lt;GP8&gt;</BrowseName>
112   <Children>
113     <Object SymbolicName="DI:ParameterSet" TypeDefinition="ua:BaseObjectType" ModellingRule="Mandatory">
114       <Children>
115         <Variable SymbolicName="FF:JointAngleS" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
116         <Variable SymbolicName="FF:JointAngleL" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
117         <Variable SymbolicName="FF:JointAngleR" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
118         <Variable SymbolicName="FF:JointAngleU" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
119         <Variable SymbolicName="FF:JointAngleB" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
120         <Variable SymbolicName="FF:JointAngleT" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory"></Variable>
121       </Children>
122     </Object>
123   </Children>
124 </ObjectType>
125
126 <ObjectType SymbolicName="FF:YRC1000" BaseType="ROB:ControllerType">
127   <BrowseName>&lt;YRC1000&gt;</BrowseName>
128 </ObjectType>
129
130 <ObjectType SymbolicName="FF:YRC1000mini" BaseType="ROB:ControllerType">
131   <BrowseName>&lt;YRC1000mini&gt;</BrowseName>
132 </ObjectType>
133
134 <ObjectType SymbolicName="FF:ConveyorIBW" BaseType="ROB:MotionDeviceType">
135   <BrowseName>&lt;ConveyorIBW&gt;</BrowseName>
136 </ObjectType>
137
138
139 <ObjectType SymbolicName="FF:SINAMICS G120C" BaseType="ROB:DriveType">
140   <BrowseName>&lt;SINAMICS G120C&gt;</BrowseName>
141   <Children>
142     <Object SymbolicName="DI:ParameterSet" TypeDefinition="ua:BaseObjectType" ModellingRule="Mandatory">
143       <Children>
144         <Variable SymbolicName="ROB:Temperature" TypeDefinition="ua:AnalogUnitType" DataType="ua:Double" ModellingRule="Mandatory">
145           <Description>The Drive temperature given by a temperature sensor inside of the Drive.</Description>
146         </Variable>
147       </Children>
148     </Object>
149   </Children>
150 </ObjectType>

```

Figure 4.4: FF Object Types in Model.xml File

The next step is to define any new Object Types, Variable Types, Data Types not covered by the imported companion specifications. The primary components of the Rocket Assembly System are 5 collaborative robot systems and a conveyor belt system. Each robot system consists of a robot arm and robot controller pair. The robot arms are composed of a mix of YASKAWA GP8s and HC10s, while the controllers are a mix of YASKAWA YRC1000s and YRC1000micro. The conveyor system is composed of 4 individual conveyor belts and 4 SINAMICS G120C VFDs. These systems include desired parameters not covered by the basic definitions provided by the Robotics companion specification, such as the joint angles of the robots. Therefore, it is necessary to create new definitions for these motion device systems specific to the Future Factories manufacturing cell.

The “HC10” Object Type is a new Object Type created in the new information model and is defined as a MotionDeviceType Object Type from the Robotics companion specification. Therefore, the HC10 Object Type inherits all of the properties and variables from the MotionDeviceType Object Type, but now additional properties and variables can be added to the definition of the new HC10 Object Type, such as the robot joint angles. The same is done for the “GP8” MotionDeviceType definition, the “YRC1000” and “YRC1000micro” ControllerType definitions for the robot controllers, the “SINAMICSG120C” DriveType definition for the conveyor VFDs, and the “IBMConveyor” MotionDeviceType for the conveyor belts. These definitions are shown in Figure 4.4.

```

141 <Object SymbolicName="FF:R01" TypeDefinition="ROB:MotionDeviceSystemType">
142 <Children>
143 <Object SymbolicName="ROB:MotionDevices" TypeDefinition="ua:FolderType" ModellingRule="Mandatory">
144 <Description>Contains any kinematic or motion device which is part of the motion device system.</Description>
145 <Children>
146 <Object SymbolicName="FF:MotionDeviceR01" TypeDefinition="FF:HC10" ModellingRule="Mandatory">
147 <BrowseName>MotionDeviceR01</BrowseName>
148 </Object>
149 </Children>
150 </Object>
151 <Object SymbolicName="ROB:Controllers" TypeDefinition="ua:FolderType" ModellingRule="Mandatory">
152 <Description>Contains the set of controllers in the motion device system.</Description>
153 <Children>
154 <Object SymbolicName="FF:ControllerR01" TypeDefinition="FF:YRC1000" ModellingRule="Mandatory">
155 <BrowseName>ControllerR01</BrowseName>
156 </Object>
157 </Children>
158 </Object>
159 <Object SymbolicName="ROB:SafetyStates" TypeDefinition="ua:FolderType" ModellingRule="Mandatory">
160 <Description>Contains safety-related data from motion device system.</Description>
161 <Children>
162 <Object SymbolicName="ROB:SafetyStateIdentifier" TypeDefinition="ROB:SafetyStateType" ModellingRule="MandatoryPlaceholder">
163 <BrowseName>&lt;SafetyStateIdentifier&gt;</BrowseName>
164 </Object>
165 </Children>
166 </Object>
167 </Children>
168 <References>
169 <Reference IsInverse="true">
170 <ReferenceType>ua:Organizes</ReferenceType>
171 <TargetId>DI:DeviceSet</TargetId>
172 </Reference>
173 </References>
174 </Object>

```

Figure 4.5: R01 Object Instance in Model.xml File

After the new Object Types for the custom information model are defined, the next step is to define the object instances for the industrial assets in the Future Factories manufacturing cell. The robot systems and the conveyor system are defined as the MotionDeviceSystemType Object Type from the Robotics companion specification. The definition for R01 is shown in Figure 4.5. The R01 system consists of a YASKAWA YRC1000 robot controller and a YASKAWA HC10 motion device. The definitions of the other motion device systems are largely similar, depending on which controllers and motion devices are used.

```

Developer PowerShell
+ Developer PowerShell
PS C:\Users\theodros\source\repos\UA-ModelCompiler\Bin\Release> .\Opc.Ua.ModelCompiler.exe -d2 C:\Users\theodros\source\repos\UA-ModelCompiler\model\OpcUaFfModel.xml -cg C:\Users\theodros\source\repos\UA-ModelCompiler\model\OpcUaFfModel.csv -o2 C:\Users\theodros\source\repos\UA-ModelCompiler\model\Published -console -version v104
Trying file: C:\Users\theodros\source\repos\UA-ModelCompiler\model\..\ModelCompiler\Design.v104\OpcUaDiModel.xml
Trying file: C:\Users\theodros\source\repos\UA-ModelCompiler\model\..\ModelCompiler\Design.v104\OpcUaDiModel.csv
Trying file: C:\Users\theodros\source\repos\UA-ModelCompiler\model\..\ModelCompiler\Design.v104\OpcUaRoboticsModel.xml
Trying file: C:\Users\theodros\source\repos\UA-ModelCompiler\model\..\ModelCompiler\Design.v104\OpcUaRoboticsModel.csv
PS C:\Users\theodros\source\repos\UA-ModelCompiler\Bin\Release>

```

Figure 4.6: Compilation of Model.xml File

rekegne, Teddy > source > repos > UA-ModelCompiler > model > Published












Name	Date modified	Type	Size
 Opc.Ua.Ff.Classes.cs	1/24/2024 5:00 PM	C# Source File	28 KB
 Opc.Ua.Ff.Constants.cs	1/24/2024 5:00 PM	C# Source File	2,318 KB
 Opc.Ua.Ff.DataTypes.cs	1/24/2024 5:00 PM	C# Source File	2 KB
 Opc.Ua.Ff.NodeIds.csv	1/24/2024 5:00 PM	Microsoft Excel C...	1 KB
 Opc.Ua.Ff.NodeSet.xml	1/24/2024 5:00 PM	XML Document	606 KB
 Opc.Ua.Ff.NodeSet2.ua	5/3/2023 12:41 PM	UA File	60 KB
 Opc.Ua.Ff.NodeSet2.xml	1/24/2024 5:00 PM	XML Document	156 KB
 Opc.Ua.Ff.PredefinedNodes.uanodes	1/24/2024 5:00 PM	UANODES File	28 KB
 Opc.Ua.Ff.PredefinedNodes.xml	1/24/2024 5:00 PM	XML Document	307 KB
 Opc.Ua.Ff.Types.bsd	1/24/2024 5:00 PM	BSD File	1 KB
 Opc.Ua.Ff.Types.xsd	1/24/2024 5:00 PM	XML Schema File	1 KB

Figure 4.7: Output of Model.xml Compilation

The design of the Model.xml file for the FF information model is now complete. The final step is to compile the Model.xml file to generate the required NodeSet2.xml, NodeIds.csv, and Types.bsd files for the building of the OPC UA server address space. This compilation process is shown in Figure 4.6. The UAModelCompiler executable is activated as indicated by “.\Opc.Ua.ModelCompiler.exe” in the terminal. The NodeIds.csv and Model.xml files from the companion specifications are passed as inputs, and based on the definitions from the Model.xml file for the FF information model, several source files are generated for the new FF namespace. The files generated as the output of the compilation process are shown in Figure 4.7. Of these files, the most important are the Opc.Ua.Ff.Types.bsd, Opc.Ua.Ff.NodeSet2.xml, and Opc.Ua.Ff.NodeIds.csv files, which will be used to build the new address space of the OPC UA server.

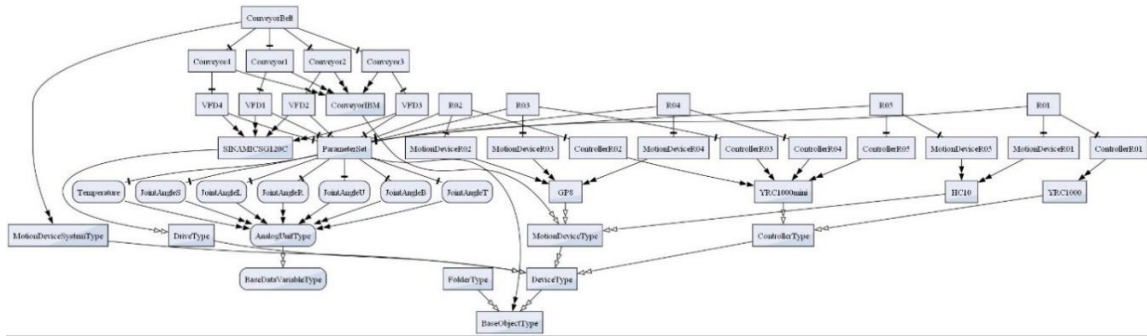


Figure 4.8: FF Information Model Diagram

A visual diagram of the full FF information model was generated using the GraphViz software. The arrows represent OPC UA references, where the black double-arrow is the “HasTypeDefinition” reference, the white double-arrow is the “HasSubtype” definition, and the bar-arrow is the “HasComponent” reference. The boxed rectangles represent Objects and Object Types, whereas the rounded rectangles represent Variables and Variable Types. The Object Types and Variable Types are indicated by the half-shading of the rectangle, whereas the non-shaded rectangles are Object and Variable instances. This graphical visualization serves to showcase the relationship between the nodes in the FF information model.

4.2.1.2 MVI STATION

Information Model

Displayname

- Objects
 - Aliases
 - Locations
 - RocketTray
 - Color
 - Config_Correct
 - Rocket_Base
 - Rocket_MidHigh
 - Rocket_MidLow
 - Rocket_Nose

NodeClass	Name	TypeDefinition	ModellingRule	Data Type		
Variable	Color	BaseDataVariabl...	No ModellingR...	String	+	×
Variable	Config_Correct	BaseDataVariabl...	No ModellingR...	Boolean	+	×
Object	Rocket_Base	BaseObjectType	No ModellingR...		+	×
Object	Rocket_MidHigh	BaseObjectType	No ModellingR...		+	×
Object	Rocket_MidLow	BaseObjectType	No ModellingR...		+	×
Object	Rocket_Nose	BaseObjectType	No ModellingR...		+	×

< Select NodeClass >

References

Reference Type	Target	
< Select Reference Type >	< Select Target >	×

Figure 4.9: RocketTray Object Instance in UaModeler

The information model for the MVI station was designed using the UAModeler software, as the MVI station is composed of only a few components, none of which require the use of domain-specific companion specifications. The building of the model in UAModeler is shown in Figure 4.9. The information model consists of the RocketTray object instance, which has an Object Type of the BaseObjectType from the OPC UA base information model. The RocketTray is made up of the 4 parts of the model rocket, which are the Rocket_Base, Rocket_MidLow, RocketMidHigh, and RocketNose object instances. The RocketTray also has 2 variable instances of the BaseDataVariableType Variable Type, which are the color of the rocket and the “Config_Correct” output of the MVI application. A visual diagram of the MVI Station information model was also generated using the GraphViz software, as shown in Figure 4.10.

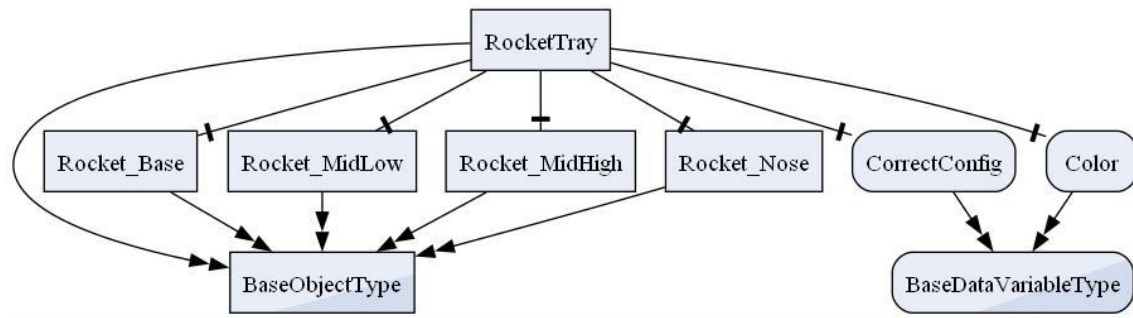


Figure 4.10: MVI Station Information Model Diagram

4.2.2 BUILDING THE OPC UA SERVER

After developing the information models to enhance the interoperability of the manufacturing cell, the next step for the PnP framework is to create the OPC UA servers for each of the AGs.

4.2.2.1 ROCKET ASSEMBLY STATION

To develop the OPC UA server for the Rocket Assembly station, 2 different methodologies are shown. The first methodology uses the internal OPC UA server from

the PLC, which is configured through the Siemens TIA Portal software. The second methodology uses a manually created OPC UA server using the open-source open62541 OPC UA stack. These methodologies serve to showcase how to build the OPC UA server for industrial machines with internal OPC UA capabilities, as well as for industrial machines without internal OPC UA capabilities, respectively. However, going forward in future sections, only the methodology using the open62541 OPC UA stack will be considered for the PnP use-case.

4.2.2.1.1 S7-1516F PLC INTERNAL OPC UA SERVER

A Siemens S7-1500 PLC is used to control the automation processes in the Future Factories manufacturing cell. A key feature of this PLC is that it contains an internal OPC UA server that can be accessed by OPC UA clients in the same network subnet. The OPC UA capabilities of the PLC can be further customized using the Siemens TIA Portal software. One such capability is the creation of server interfaces in TIA Portal. Server interfaces combine nodes of an OPC UA address space into a single unit, so that individual server interfaces can be viewed by OPC UA clients. A server interface contains nodes that can be read with an OPC UA client to receive tag data from the machine, nodes that you can write to with an OPC UA client to transfer new values to the data tags of the machine, and nodes that can be called with an OPC UA client to start functions of the machine using server methods.

12_18_2023_TeddyOPCUA_V18 > USC_PLC [CPU 1516F-3 PN/DP] > OPC UA communication > Server interfaces > Reference namespaces > Opc.Ua.Di.NodeSet2

Export interfaceConsistency checkUpdate interfaceImport companion specification

OPC UA server interface

Browse name	Node type	Local data	Data type
1 OpC.Ua.Di.NodeSet2	Reference node set		
2 http://opcfoundation.org/UA/DI/	Namespace		
3 Data types	Folder		
4 FetchResultDataType	FetchResultDataType		
5 TransferResultErrorDataType	TransferResultError...		
6 TransferResultDataDataType	TransferResultData...		
7 ParameterResultDataType	ParameterResultDat...		
8 UpdateBehavior	UpdateBehavior		
9 Object types	Folder		
10 TopologyElementType	TopologyElementT...		
11 IVendorNameplateType	IVendorNameplate...		
12 ITagNameplateType	ITagNameplateType		
13 IDeviceHealthType	IDeviceHealthType		
14 ISupportInfoType	ISupportInfoType		
15 ComponentType	ComponentType		
16 DeviceType	DeviceType		
17 SoftwareType	SoftwareType		
18 BlockType	BlockType		
19 DeviceHealthDiagnosticAlarm...	DeviceHealthDiagn...		
20 FailureAlarmType	FailureAlarmType		

OPC UA elements

Project data	Data type
1 Software units	
2 Program blocks	
3 Actuators	
4 AutoCellControl	AutoCellControl
5 Conveyors	
6 EdgeMonitoring	EdgeMonitoring
7 HMI	
8 Move_Data	Move_Data
9 OpcenterCellControl	OpcenterCellControl
10 OPCUA_Test	OPCUA_Test
11 Robotic Signals	
12 Safety	
13 PLC data types	
14 LDnSafe_typeSinaGtlg30Control	LDnSafe_typeSina...
15 LDnSafe_typeSinaGtlg305status	LDnSafe_typeSina...
16 Telegramin	Telegramin
17 TelegramOut	TelegramOut
18 Technology objects	

Figure 4.11: DI Reference Namespace Server Interface

12_18_2023_TeddyOPCUA_V18 > USC_PLC [PCU 1516F-3 PN/DP] > OPC UA communication > Server interfaces > Reference namespaces > Opc.Ua.Robotics.NodeSet2

Export interface

Consistency check

Update interface

Import companion specification

OPC UA server interface

Browse name	Node type	Local data	Data type
OpC.Ua.Robotics.NodeSet2	Reference node set		
http://opcfoundation.org/UA/Robotics/	Namespace		
Data types	Folder		
Object types	Folder		
MotionDeviceSystemType	MotionDeviceSyste...		
MotionDevices	Folder		
Controllers	Folder		
SafetyStates	Folder		
MotionDeviceType	MotionDeviceType		
ParameterSet	Object		
Axes	Folder		
PowerTrains	Folder		
FlangeLoad	Object		
AdditionalComponents	Folder		
Manufacturer	LocalizedText		
Model	LocalizedText		
ProductCode	WSTRING		
SerialNumber	WSTRING		
MotionDeviceCategory	MotionDeviceCateg...		
AxisType	AxisType		
PowerTrainType	PowerTrainType		

OPC UA elements

Project data	Data type
Software units	
Program blocks	
PLC data types	
Technology objects	

Figure 4.12: Robotics Reference Namespace Server Interface

TIA Portal categorizes OPC UA server interfaces into either the server interface type or the companion specification type. Each server interface serves to define one or more namespaces in the OPC UA server. The server interface is used to add OPC UA elements from the PLC address space directly as the nodes of the OPC UA server, while the companion specification type defines the OPC UA server address space according to the nodes of an imported OPC UA information model. Furthermore, companion specification server interfaces are classified into 2 types: companion specifications and reference namespaces. The companion specification type allows for OPC UA elements, such as Data Block variables and PLC tags, to be mapped to specific nodes of the

information model. The reference namespace type imports the definition of object, variable, and data types that can be utilized by the companion specification server interface.

Because the FF information model is dependent on type definitions from the DI and Robotics companion specifications, it was necessary to add the DI and Robotics information models as reference namespaces to the TIA Portal Project, as shown in Figures 4.11 and 4.12. In total, the S7-1516F OPC UA server has 3 individual server interfaces, where the DI server interface depends on the OPC UA base information model, the Robotics server interface depends on the DI server interface, and the FF server interface depends on both the DI and Robotics server interfaces.

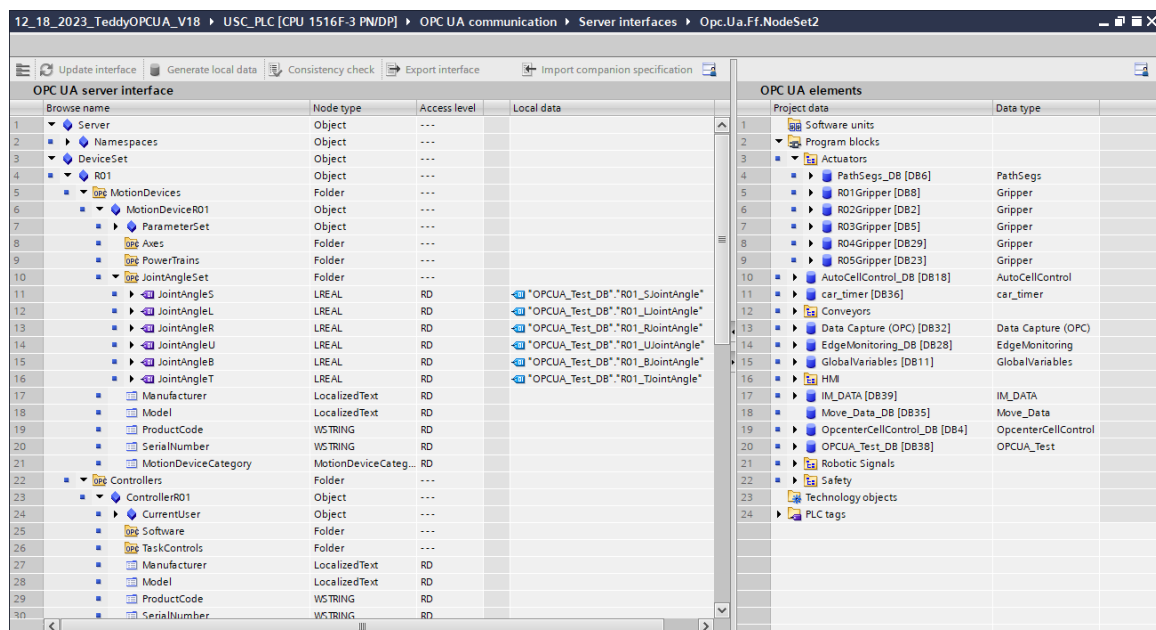


Figure 4.13: FF Companion Specification Server Interface

The developed FF information model for the manufacturing cell was uploaded to TIA Portal as an OPC UA companion specification server interface called “FF.” Figure 4.13 shows the developed FF companion specification server interface for the custom information model. It is important to note that TIA Portal refers to the OPC UA companion specifications as reference namespaces and refers to the information model that uses these

reference namespaces as companion specifications, even though the developed information model is not technically an OPC UA companion specification. This is because custom OPC UA information models in TIA Portal are usually developed as the server interface type instead of the companion specification type. Since the information model was developed using the UA Model Compiler tool, there was no need to recreate the information model in TIA Portal.

However, it is important to note that there are certain disadvantages to using this methodology. The PLC includes several features allowing for the communication between industrial machines; however, this assumes that the device connected to the PLC is supported, which is indicated by the presence of the device within the TIA Portal hardware catalog. For devices and systems that exist outside the hardware catalog, such as a Raspberry Pi, it becomes difficult to enable seamless communication without a large manual configuration effort from the system integrator. Even with such an effort, the configuration effort is typically different depending on what drivers and modules are necessary to enable communication between 2 specific devices. For example, the robot controller has several parameters that describe the state of the robot and the controller, but most of these parameters cannot be directly accessed by the PLC. The PnP middleware is designed to be generic as to expose the manufacturing data to both industrial assets and IT systems alike in a scalable manner. As such is the case, this research work opts to use the 2nd methodology to acquire data from the PLC into an OPC UA server.

4.2.2.1.2 OPEN62541 OPC UA SERVER

An OPC UA server was manually configured using the open62541 library on a Raspberry Pi 4 running Ubuntu 20.04.5, serving as the AG for the Rocket Assembly

station. The open62541 library is an open-source implementation of OPC UA written in the C language. The library includes a nodeset compiler tool that creates an internal representation of the information model and generates the associated C source code for each namespace. This internal representation is defined according to the CSV (Node IDs), BSD (OPC UA Types), and XML (Nodeset) files generated from the UAModelCompiler tool.

```

11  □ua_generate_nodeset_and_datatypes(
12      ..... NAME "di"
13      ..... TARGET_PREFIX "${PROJECT_NAME}"
14      ..... FILE_CSV "${PROJECT_SOURCE_DIR}/nodesets/DI/OpcUaDiModel.csv"
15      ..... FILE_BSD "${PROJECT_SOURCE_DIR}/nodesets/DI/Opc.Ua.Di.Types.bsd"
16      ..... OUTPUT_DIR "${GENERATE_OUTPUT_DIR}"
17      ..... FILE_NS "${PROJECT_SOURCE_DIR}/nodesets/DI/Opc.Ua.Di.NodeSet2.xml"
18      ..... INTERNAL
19      ..... DEPENDS "${UA_FILE_NS0}"
20  )
21  □ua_generate_nodeset_and_datatypes(
22      ..... NAME "rob"
23      ..... TARGET_PREFIX "${PROJECT_NAME}"
24      ..... FILE_CSV "${PROJECT_SOURCE_DIR}/nodesets/Robotics/NodeIds.csv"
25      ..... FILE_BSD "${PROJECT_SOURCE_DIR}/nodesets/Robotics/Opc.Ua.Robotics.Types.bsd"
26      ..... OUTPUT_DIR "${GENERATE_OUTPUT_DIR}"
27      ..... FILE_NS "${PROJECT_SOURCE_DIR}/nodesets/Robotics/Opc.Ua.Robotics.NodeSet2.xml"
28      ..... INTERNAL
29      ..... DEPENDS "${UA_FILE_NS0}" "di"
30  )
31  □ua_generate_nodeset_and_datatypes(
32      ..... NAME "ff"
33      ..... TARGET_PREFIX "${PROJECT_NAME}"
34      ..... FILE_CSV "${PROJECT_SOURCE_DIR}/model/OpcUaFfModel.csv"
35      ..... FILE_BSD "${PROJECT_SOURCE_DIR}/model/Published/Opc.Ua.Ff.Types.bsd"
36      ..... OUTPUT_DIR "${GENERATE_OUTPUT_DIR}"
37      ..... FILE_NS "${PROJECT_SOURCE_DIR}/model/Published/Opc.Ua.Ff.NodeSet2.xml"
38      ..... INTERNAL
39      ..... DEPENDS "${UA_FILE_NS0}" "di" "rob"
40  )
41  # Previous macro automatically sets some variables which hold the generated source code files using the provided NAME
42  □add_executable(main main.c
43      ..... ${UA_NODESSET_DI_SOURCES}
44      ..... ${UA_TYPES_DI_SOURCES}
45      ..... ${UA_NODESSET_ROB_SOURCES}
46      ..... ${UA_TYPES_ROB_SOURCES}
47      ..... ${UA_NODESSET_FF_SOURCES}
48      ..... ${UA_TYPES_FF_SOURCES}
49      ..... )
50  # Make sure the nodeset compiler is execute before compiling the main file
51  □add_dependencies(main
52      ..... ${PROJECT_NAME}-ns-di
53      ..... ${PROJECT_NAME}-ns-rob
54      ..... ${PROJECT_NAME}-ns-ff
55      ..... )
56  target_link_libraries(main open62541::open62541)

```

Figure 4.14: Generate FF Namespaces and Data Types

A CMakeLists.txt file was created to hold the CMake functions “generate_nodeset_and_datatypes” that call the nodeset compiler tool. A code snippet of the CMakeLists.txt file is shown in Figure 4.14. Using these CMake functions, the nodeset compiler tool generates the source code for the namespace and data types for each

information model. The source code is then linked to the main function for the OPC UA server.

```

1352  /* R01 - ns=3;i=20857 */
1353
1354  static UA_StatusCode function_namespace_ff_generated_67_begin(UA_Server *server, UA_UInt16* ns) {
1355      UA_StatusCode retVal = UA_STATUSCODE_GOOD;
1356      UA_ObjectAttributes attr = UA_ObjectAttributes_default;
1357      attr.eventNotifier = UA_EVENTNOTIFIER_SUBSCRIBE_TO_EVENT;
1358      attr.displayName = UA_LOCALIZEDTEXT("", "R01");
1359      retVal |= UA_Server_addNode_begin(server, UA_NODECLASS_OBJECT, UA_NODEID_NUMERIC(ns[3], 20857LU),
1360      UA_NODEID_NUMERIC(ns[1], 5001LU), UA_NODEID_NUMERIC(ns[0], 35LU), UA_QUALIFIEDNAME(ns[3], "R01"),
1361      UA_NODEID_NUMERIC(ns[2], 1002LU), (const UA_NodeAttributes*)&attr, &UA_TYPES[UA_TYPES_OBJECTATTRIBUTES], NULL, NULL);
1362      return retVal;
1363  }
1364
1365  static UA_StatusCode function_namespace_ff_generated_67_finish(UA_Server *server, UA_UInt16* ns) {
1366      return UA_Server_addNode_finish(server, UA_NODEID_NUMERIC(ns[3], 20857LU));
1367  }
1368

```

Figure 4.15: R01 Object Node in namespace_ff_generated.c File

```

14  #define UA_FFID_HC10_PARAMETERSSET_JOINTANGLES 15001 /* Variable */
15  #define UA_FFID_CONVEYORBELT 15002 /* Object */
16  #define UA_FFID_CONVEYORBELT_PARAMETERSSET 15003 /* Object */
17  #define UA_FFID_CONVEYORBELT_PARAMETERSSET_PARAMETERIDENTIFIER 15004 /* Variable */
18  #define UA_FFID_CONVEYORBELT_METHODSET 15005 /* Object */
19  #define UA_FFID_CONVEYORBELT_METHODSET_METHODIDENTIFIER 15006 /* Method */

```

Figure 4.16: FF Node IDs in ff_nodeids.h File

As a result, several source files are generated for each information model. These source files are the types_x_generated.c and namespace_x.c source files, as well as the associated types_x_generated.h and namespace_x.h header files, where “x” represents the name of the information model i.e., di, rob, and ff. An x_nodeids.h header file is also generated for each information model and contains the definitions for the Node IDs for each node in the server address space. A code snippet of the namespace_ff_generated.c file is displayed in Figure 4.15, showing the definition of the R01 object node in the information model. This file includes the definition of the all of the nodes in the OPC UA server address space. A code snippet of the ff_nodeids.h file is shown in Figure 4.16. This file defines the Node IDs for each node in the information model. There are no unique data types for the custom information model, so the types_ff_generated.h file is not shown.

```

18  static volatile UA_Boolean running = true;
19
20  static void stopHandler(int sig) {
21      UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_USERLAND, "received ctrl-c");
22      running = false;
23  }
24  // - Main Server
25
26  int main(int argc, char** argv) {
27      signal(SIGINT, stopHandler);
28      signal(SIGTERM, stopHandler);
29
30      UA_Server *server = UA_Server_new();
31      UA_ServerConfig_setDefault(UA_Server_getConfig(server));
32
33      UA_StatusCode retval;
34      /* create nodes from nodeset */
35      if(namespace_di_generated(server) != UA_STATUSCODE_GOOD) {
36          UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Could not add the DI nodeset."
37              "Check previous output for any error.");
38          UA_Server_delete(server);
39          retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
40      }
41      if(namespace_rob_generated(server) != UA_STATUSCODE_GOOD) {
42          UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Could not add the Robotics nodeset."
43              "Check previous output for any error.");
44          UA_Server_delete(server);
45          retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
46      }
47      if(namespace_ff_generated(server) != UA_STATUSCODE_GOOD) {
48          UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Could not add the FF nodeset."
49              "Check previous output for any error.");
50          retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
51      }
52      else {
53          retval = UA_Server_run(server, &running);
54      }
55
56      UA_Server_delete(server);
57      return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
58  }

```

Figure 4.17: FF Base Server Code From main.c File

The main.c file contains the code for building the OPC UA address space and running the server. The base version of the server code before the server nodes are populated with data is shown in Figure 4.17. The header files for the namespaces, node ids, and data types are included into the server code to create the custom address space for the OPC UA server. The status of the namespace for each information model is checked for any issues before it is uploaded to the server address space.

4.2.2.2 MOBILE VISUAL INSPECTION STATION OPC UA SERVER

The process of building the OPC UA server for the MVI station is largely similar to the building process for the open62541 FF OPC UA server. The OPC UA server was

manually configured using the open62541 library on another Raspberry Pi 4 running Ubuntu 22.04.5, serving as the AG for the MVI station.

```

12  ua_generate_nodeset_and_datatypes(
13      .....NAME "mvi"
14      .....TARGET_PREFIX "${PROJECT_NAME}"
15      .....OUTPUT_DIR "${GENERATE_OUTPUT_DIR}"
16      .....FILE_NS "${PROJECT_SOURCE_DIR}/model_mvi/Opc.Ua.Mvi.NodeSet2.xml"
17      .....INTERNAL
18      .....DEPENDS "${UA_FILE_NS0}"
19  )
20  # Previous macro automatically sets some variables which hold the generated source code files using the provided NAME
21  add_executable(main main.c .....
22      .....${UA_NODESSET_MVI_SOURCES}
23      .....${UA_TYPES_MVI_SOURCES}
24      .....
25  )
26  # Make sure the nodeset compiler is execute before compiling the main file
27  add_dependencies(main
28      .....${PROJECT_NAME}-ns-mvi
29      .....
30  )
31  target_link_libraries(main open62541::open62541)

```

Figure 4.18: Generate MVI Namespace from CMakeLists.txt File

A CMakeLists.txt file was created to hold the CMake function that calls the nodeset compiler tool. Because the information model for the MVI station does not use any companion specifications, the nodeset compiler tool does not require a NodeIds.csv or Types.bsd file. A code snippet of the CMakeLists.txt file is shown in Figure 4.18. The nodeset compiler tool then generates the source code for the namespace of the information model.

```

211  static UA_StatusCode function_namespace_mvi_generated_9_begin(UA_Server *server, UA_UInt16* ns) {
212      UA_StatusCode retVal = UA_STATUSCODE_GOOD;
213      UA_VariableAttributes attr = UA_VariableAttributes_default;
214      attr.minimumSamplingInterval = 0.000000;
215      attr.userAccessLevel = 1;
216      attr.accessLevel = 3;
217      /* Value rank inherited */
218      attr.valueRank = -2;
219      attr.dataType = UA_NODEID_NUMERIC(ns[0], 1LU);
220      attr.displayName = UA_LOCALIZEDTEXT("", "Config_Correct");
221      retVal |= UA_Server_addNode_begin(server, UA_NODECLASS_VARIABLE, UA_NODEID_NUMERIC(ns[1], 6003LU),
222      UA_NODEID_NUMERIC(ns[1], 5006LU), UA_NODEID_NUMERIC(ns[0], 47LU), UA_QUALIFIEDNAME(ns[1], "Config_Correct"),
223      UA_NODEID_NUMERIC(ns[0], 63LU), (const UA_NodeAttributes*)&attr, &UA_TYPES[UA_TYPES_VARIABLEATTRIBUTES], NULL, NULL);
224      return retVal;
225  }
226
227  static UA_StatusCode function_namespace_mvi_generated_9_finish(UA_Server *server, UA_UInt16* ns) {
228      return UA_Server_addNode_finish(server, UA_NODEID_NUMERIC(ns[1], 6003LU));
229  }

```

Figure 4.19: Config_Correct Variable Node in namespace_mvi_generated.c File

Because there are no unique Object Types, Variable Types, Data Types, or Reference Types created for the MVI Station information model, the only output of the nodeset compiler tool is the namespace_mvi_generated.c and namespace_mvi_generated.h

files. The namespace_mvi_generated.c file is displayed in Figure 4.19. The source code is then linked to the main function for the OPC UA server.

```

22  static volatile UA_Boolean running = true;
23
24  static void stopHandler(int sign) {
25      UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
26      running = false;
27  }
28
29  int main(int argc, char** argv) {
30      signal(SIGINT, stopHandler);
31      signal(SIGTERM, stopHandler);
32
33      UA_Server *server = UA_Server_new();
34      UA_ServerConfig_setDefault(UA_Server_getConfig(server));
35
36      UA_StatusCode retval;
37      /* create nodes from nodeset */
38      if(namespace_mvi_generated(server) != UA_STATUSCODE_GOOD) {
39          UA_LOG_ERROR(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "Could not add the MVI nodeset.");
40          "Check previous output for any error.";
41          UA_Server_delete(server);
42          retval = UA_STATUSCODE_BADUNEXPECTEDERROR;
43      }
44      else {
45          retval = UA_Server_run(server, &running);
46      }
47
48      UA_Server_delete(server);
49      return retval == UA_STATUSCODE_GOOD ? EXIT_SUCCESS : EXIT_FAILURE;
50  }

```

Figure 4.20: MVI Base Server Code from main.c File

Following the same steps as for the Rocket Assembly station, the main.c file for the MVI station is created. The base version of the server code before the server nodes are populated with data is shown in Figure 4.20. The status of the namespace for the MVI information model is checked for any issues before it is uploaded to the server address space.

4.2.3 DATA ACQUISITION AND POPULATION OF THE OPC UA SERVER

Once the server address space has been configured, the next step is to populate the nodes of the server address spaces with process data. This step is highly variable depending on the choice of communication protocol and the method by which the device publishes the data. For OPC UA connections – meaning an OPC UA client/server connection – the OPC UA specification includes numerous protocols such as the ultra-fast OPC-binary

transport, JSON over web-sockets, etc. The optional OPC UA PubSub extension is protocol agnostic and can be used with broker-based protocols like MQTT and AMQP, as well as broker-less implementations like UDP-Multicasting, with the caveat that published messages follow the content structure for PubSub messages defined by the OPC Foundation. However, it becomes slightly more difficult to get data into the OPC UA server when the publishing device does not use the OPC UA framework, as is often the case when collecting data from legacy machines. The published data must be translated into a format compatible with the OPC UA architecture to be added to the OPC UA server. To demonstrate such a use-case, the Future Factories manufacturing process data is published using the widely compatible MQTT protocol and translated into the OPC UA architecture.

4.2.3.1 ROCKET ASSEMBLY STATION

The Future Factories cell uses a Siemens IPC 227E Industrial PC acting as an edge device to collect tag information from a S7-1516F PLC through the proprietary S7 Connection protocol. The SIMATIC S7 Connector is an application that runs on the edge device that accesses the PLC data through either an S7 Connection or OPC UA. The application sends the PLC Tag values imported from the controllers to the Industrial Edge (IE) Databus via the Industrial Edge Runtime. A data publisher application was created on the edge device to parse through data from the IE Databus for the desired tags and publish the filtered MQTT topic to an MQTT broker.

```
24 void on_message(struct mosquitto *mosq, void *obj, const struct mosquitto_message *msg) {
25
26     //printf("New message with topic %s: %s\n", msg->topic, (char *) msg->payload);
27     char *data = (char *) msg->payload;
28     cJSON *root = cJSON_Parse(data);
29     if (root == NULL) {
30         fprintf(stderr, "Error parsing JSON: %s\n", cJSON_GetErrorPtr());
31     }
32     cJSON *VFD1_Temperature = cJSON_GetObjectItemCaseSensitive(root, "Q_VFD1_Temperature");
```

Figure 4.21: Parsing FF Data in MQTT Client

```

63     key_t key = ftok("shared_memory_key", 1);
64     int shmid = shmget(key, sizeof(float), IPC_CREAT | 0666);
65     float *tags_ptr = (float *)shmat(shmid, NULL, 0);
66
67     for(int i=0; i < 28; i++){
68         if(i==0){
69             tags[i] = VFD1_Temperature->valuedouble;
70             tags_ptr[i] = tags[i];
71         }

```

Figure 4.22: FF Data Assignment in MQTT Client

An MQTT client application was then created on the AG to subscribe to the topic from the MQTT broker at a regular time interval of 500 milliseconds. The message payload data is received in the form of a JSON object which is assigned to the char variable “data.” Using the lightweight cJSON library, the data variable is parsed for each index within the JSON object, and each index is assigned to a new variable that corresponds with the tag name, as shown in Figure 4.21. A shared memory segment is then created in the MQTT client application. A shared memory segment is an area of memory in a computer system that can be accessed by multiple processes concurrently. By utilizing this shared memory segment, the data assigned to the shared memory segment in the MQTT client can then be accessed by the OPC UA server application. A new double array variable “tags[]” is declared. The tag variables are then assigned to indices within the tags[] array. The entire tags[] array is then assigned to the variable stored in the shared memory segment, which is called “tags_ptr.” This process is shown in Figure 4.22.

```

31 static void beforeRead_Q_VFD1_Temperature(UA_Server *server, const UA_NodeId *sessionId, void *sessionContext,
32     const UA_NodeId *nodeid, void *nodeContext, const UA_NumericRange *range, const UA_DataValue *data) {
33     key_t key = ftok("shared_memory_key", 1);
34     int shmid = shmget(key, sizeof(float), IPC_CREAT | 0666);
35     float *tags_ptr = (float *)shmat(shmid, NULL, 0);
36     int i=0;
37     UA_Double Q_VFD1_Temperature = tags_ptr[i];
38     ...
39     UA_Variant value;
40     UA_Variant_setScalarCopy(&value, &Q_VFD1_Temperature, &UA_TYPES[UA_TYPES_DOUBLE]);
41     UA_Server_writeValue(server, UA_NODEID_NUMERIC(4, 17552), value);
42     shmdt(tags_ptr);
43 }

```

Figure 4.23: Callback Function for Q_VFD1_Temperature Variable Node

```

756     else {
757
758         //Tag Callbacks
759         UA_ValueCallback callback_Q_VFD1_Temperature;
760         callback_Q_VFD1_Temperature.onRead = beforeRead_Q_VFD1_Temperature;
761         callback_Q_VFD1_Temperature.onWrite = NULL;
762         UA_Server_setVariableNode_valueCallback(server, UA_NODEID_NUMERIC(4,17552), callback_Q_VFD1_Temperature);
763     }

```

Figure 4.24: Calling beforeRead_Q_VFD1_Temperature Callback Function

Now that the MQTT client has been completed, the main.c file for the OPC UA server application must be modified to populate the server nodes with data. Using the shared memory segment created in the MQTT client application, the OPC UA server application can access the published data from the MQTT client application and write to the value attribute of the appropriate OPC UA node through a callback mechanism. A callback function is created for each node in the OPC UA address space. Figure 4.23 defines the callback function for the Q_VFD1_Temperature variable node, which is named “beforeRead_Q_VFD1_Temperature.” Lines 33-37 retrieves the “tags_ptr” array from the MQTT Client and assigns the value to a new variable of the associated OPC UA Data Type i.e., UA_Double for Q_VFD1_Temperature. Lines 39-41 uses the open62541 API to write to the value attribute of the Q_VFD1_Temperature variable node in the OPC UA address space, utilizing the OPC UA attribute service set described in Chapter 3. The callback function is then called in the main loop of the main.c server code through a callback mechanism, as shown in Figure 4.24. Calling the callback function triggers the OPC UA write service to write the updated value metric from the MQTT client to the value attribute of the OPC UA server node.

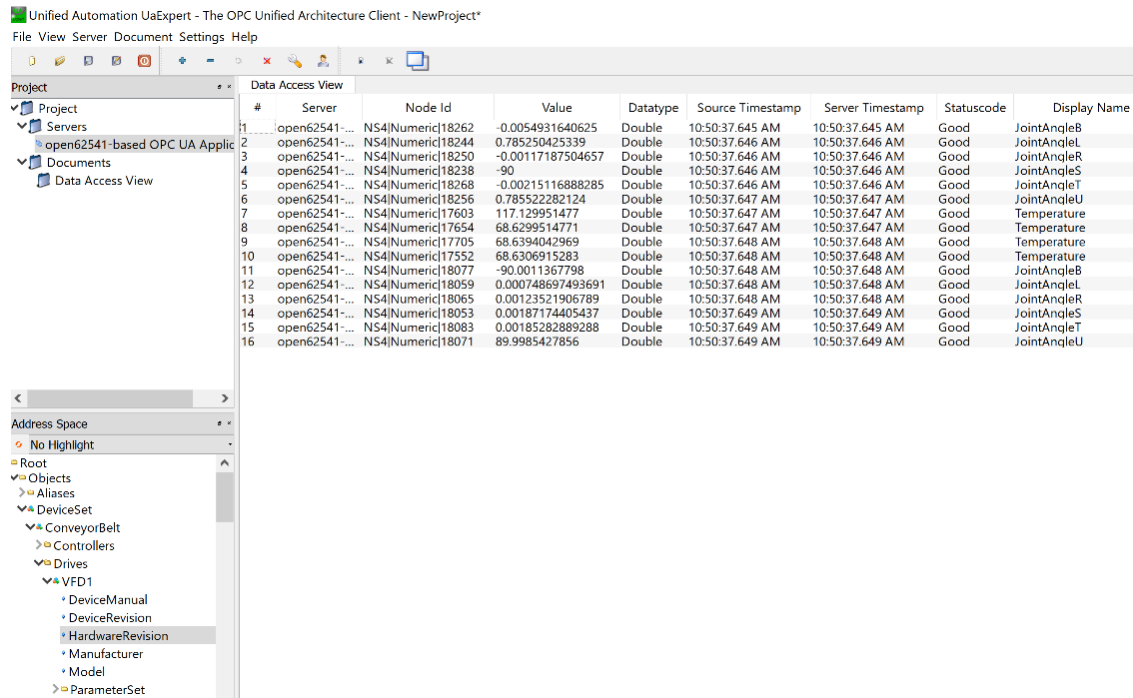


Figure 4.25: UAExpert Client Connection to FF OPC UA Server

Once an OPC UA client reads the node from the OPC UA server, the value attribute will then be updated from the data received by the MQTT client. Figure 4.25 shows the connection of the UAExpert OPC UA client to the FF OPC UA server, indicating that the server nodes are now populated with data. The configuration of the AG for the Rocket Assembly station has now been completed. The final step is to connect the FF AG to the same network as the IG to enable access to the manufacturing data.

4.2.3.2 MOBILE VISUAL INSPECTION STATION

The same process is repeated for the population of the OPC UA server nodes for the MVI station with process data. An MQTT client is created to access the process data from the MVI Station MQTT broker.

```

33  cJSON *correctconfig = cJSON_GetObjectItemCaseSensitive(root, "Config_Correct");
34
35
36  if (cJSON_IsTrue(correctconfig)){
37
38      key_t key = ftok("shared_memory_key", 1);
39      int shmid = shmget(key, sizeof(bool), IPC_CREAT | 0666);
40      bool *correctconfig_ptr = (bool *)shmat(shmid, NULL, 0);
41      correctconfig_ptr = correctconfig;
42      shmdt(correctconfig_ptr);
43  }
44  else if (cJSON_IsFalse(correctconfig)){
45      key_t key = ftok("shared_memory_key", 1);
46      int shmid = shmget(key, sizeof(bool), IPC_CREAT | 0666);
47      bool *correctconfig_ptr = (bool *)shmat(shmid, NULL, 0);
48      correctconfig_ptr = correctconfig;
49      shmdt(correctconfig_ptr);
50  }

```

Figure 4.26: Parsing MVI Data in MQTT Client

The message payload data is parsed for each index within the JSON object, and each index is assigned to a new variable that corresponds with the tag name, as shown for the “ConfigCorrect” variable in Figure 4.26. The shared memory segment is then created in the MQTT client application, and the value of the “ConfigCorrect” variable is assigned to the shared memory segment to be accessed from the OPC UA server application.

```

26  static void beforeRead_ConfigCorrect(UA_Server *server, const UA_NodeId *sessionId, void *sessionContext,
27  const UA_NodeId *nodeid, void *nodeContext, const UA_NumericRange *range, const UA_DataValue *data) {
28      key_t key = ftok("shared_memory_key", 1);
29      int shmid = shmget(key, sizeof(bool), IPC_CREAT | 0666);
30      bool *correctconfig_ptr = (bool *)shmat(shmid, NULL, 0);
31      UA_Boolean ConfigCorrect = *correctconfig_ptr;
32
33      UA_Variant value;
34      UA_Variant_setScalarCopy(&value, &ConfigCorrect, &UA_TYPES[UA_TYPES_BOOLEAN]);
35      UA_Server_writeValue(server, UA_NODEID_NUMERIC(2, 6003), value);
36      shmdt(correctconfig_ptr);
37  }

```

Figure 4.27: Callback Function for ConfigCorrect Variable Node

```

62  else {
63      //Tag Callbacks
64      UA_ValueCallback callback_ConfigCorrect;
65      callback_ConfigCorrect.onRead = beforeRead_ConfigCorrect;
66      callback_ConfigCorrect.onWrite = NULL;
67      UA_Server_setVariableNode_valueCallback(server, UA_NODEID_NUMERIC(2, 6003), callback_ConfigCorrect);
68
69      retval = UA_Server_run(server, &running);
70  }

```

Figure 4.28: Calling beforeRead_ConfigCorrect Callback Function

Now that the MQTT client is complete, the main.c file for the OPC UA server application is modified to populate the server nodes with data. A callback function is created for each node in the OPC UA address space. Figure 4.27 defines the callback function for the ConfigCorrect variable node, which is named “beforeRead_ConfigCorrect.” The callback function is then called in the main loop of the main.c server code through the callback mechanism, as shown in Figure 4.28.

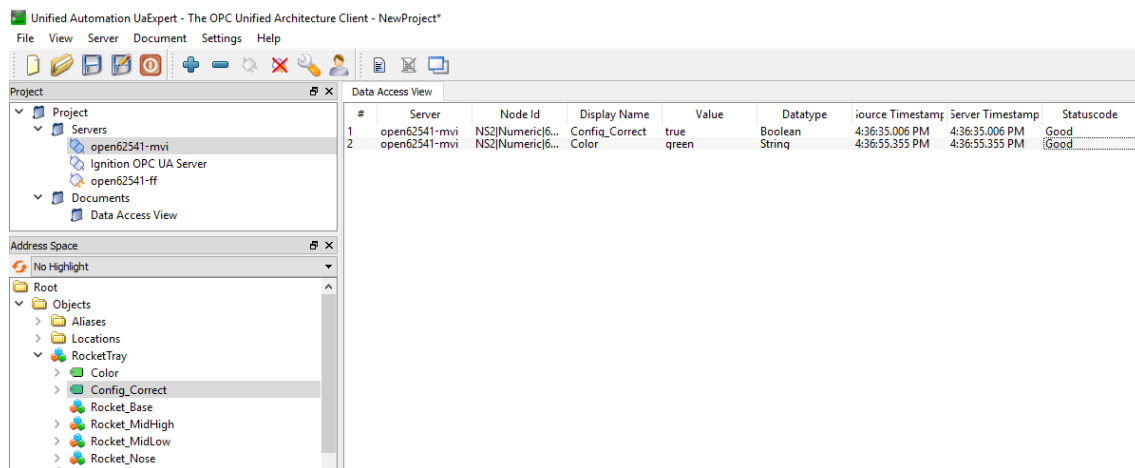


Figure 4.29: UAExpert Client Connection to MVI OPC UA Server

The value attribute is updated from the data received by the MQTT client upon an OPC UA client connection. Figure 4.29 shows the connection of the UAExpert OPC UA client to the MVI station OPC UA server, indicating that the server nodes are now populated with data. The configuration of the AG for the MVI station is now complete. The final step is to connect the MVI station AG to the same network as the IG to enable access to the manufacturing data.

4.3 IGNITION GATEWAY

After the AGs have been configured, the IG must be developed. The Ignition software, as well as the Cirrus Link MQTT Distributor, Engine, and Transmission modules, were installed on an NVidia Jetson TX2 running the Ubuntu 18.04 operating system. The

device serves as the central gateway in the PnP architecture. This section describes the configuration process of the IG.

4.3.1 CONFIGURE TAG PROVIDERS AND CONNECT AGENT OPC UA SERVERS

Name	Description	Enabled	Type	
FF		true	Standard Tag Provider	delete edit
MVI		true	Standard Tag Provider	delete edit

[→ Create new Realtime Tag Provider...](#)

Figure 4.30: Ignition Tag Providers for AGs

Name	Type	Description	Read Only	Status	
Ignition OPC UA Server	OPC UA	A "loopback" connection to the Ignition OPC UA server running on this gateway.	false	Connected	More ▾ edit
open62541_ff	OPC UA		false	Connected	More ▾ edit
open62541_mvi	OPC UA		false	Connected	More ▾ edit

[→ Create new OPC Connection...](#)

Note: For details about a connection's status, see the [OPC Connection Status](#) page.

Figure 4.31: Ignition OPC Connections for AGs

A standard Tag Provider was created for each AG for the collection of the manufacturing data. “FF” is the Tag Provider that will contain the data from the Rocket Assembly station, whereas “MVI” is the Tag Provider that will contain the data from the MVI station. These Tag Providers are shown in Figure 4.30. In the PnP architecture, the Tag Providers essentially act as clients for the AG OPC UA servers, consuming the manufacturing data and translating the data into Ignition Tags. An OPC connection was then created for each of the AG OPC UA servers to the IG, as shown in Figure 4.31. This step exposes the OPC UA servers to the IG, allowing for the consumption of the manufacturing data from the AGs.

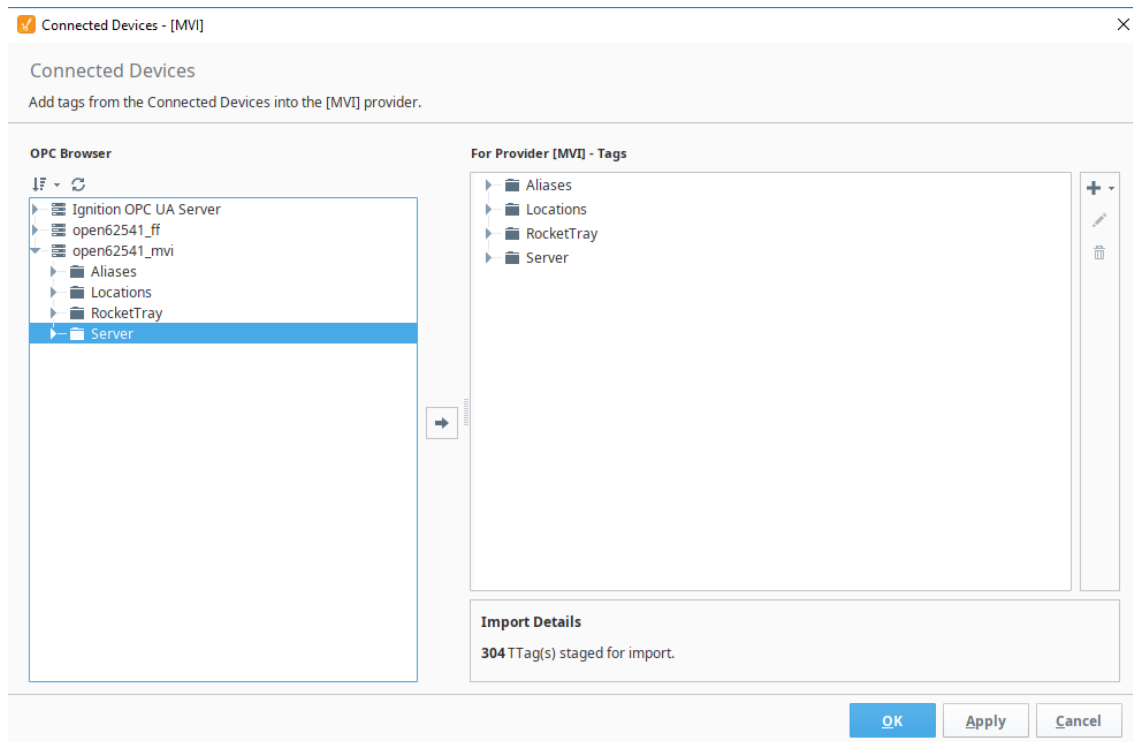


Figure 4.32: Adding MVI Agent to Tag Provider

In the Ignition Designer application, the Tag Browser tool is used to interact with the Tag Providers in the IG. After selecting the Tag Provider, the data sources connected to the IG were then browsed. The data sources were then imported into the Tag Provider for the respective AG. Figure 4.32 shows the importing of the MVI Agent OPC UA server nodes into the MVI Tag Provider.

TYPE	ACTION	TITLE
Server	refresh	[-] Ignition OPC UA Server
Object		[-] Devices
Object		[-] Server
Object		[-] Tag Providers
Object		[-] FF
Object		[-] MQTT Distributor
Object		[-] MQTT Engine
Object		[-] MQTT Transmission
Object		[-] MVI
Object		[-] System

Figure 4.33: Client Connection to Ignition OPC UA Server

Navigating to the OPC UA Server Settings within the IG webserver, the “Expose Tag Providers” option was enabled. This setting allows users to bridge the Tag Providers to the local Ignition OPC UA Server. Clients now have access to data across the entire manufacturing ecosystem via a single OPC UA server-client connection. Figure 4.33 shows an OPC UA client connection to the Ignition OPC UA server.

4.3.3 CONFIGURE CIRRUS LINK MQTT MODULES

The manufacturing data from the AGs is now exposed to the Ignition OPC UA Server. The next step is to configure the Cirrus Link MQTT Modules to translate between the OPC UA framework and the MQTT Sparkplug IIoT Protocol. This step involves the configuration of the MQTT Distributor, Engine, and Transmission modules.

4.3.3.1 MQTT DISTRIBUTOR

General	
Main	
Enabled	<input checked="" type="checkbox"/> Enable the MQTT Server
Non-TLS Settings	
Enable TCP	<input checked="" type="checkbox"/> Enable plain TCP connections for the MQTT Server
Port	1883 Non-TLS MQTT Server port
Enable Websocket	<input checked="" type="checkbox"/> Enable Websocket connections for the MQTT Server
Websocket Port	8090 Non-TLS MQTT Server Websocket port

Figure 4.34: MQTT Distributor Module Configuration

The MQTT Distributor module is configured to create a local MQTT Server on port 1883 and 8090 of the IG, as shown in Figure 4.34. The server acts as a local MQTT broker for MQTT clients on the same network to publish and subscribe data. Port 1883 serves as the device port for MQTT-over-TCP connections, whereas port 8090 serves as the device port for MQTT-over-Websocket connections.

4.3.3.2 MQTT TRANSMISSION

A new server set called “Chariot SCADA Sparkplug” was configured in the MQTT Transmission module. Server sets represent a logical grouping of MQTT servers that are referred to by MQTT Transmitters.

General	Servers	Sets	Transmitters	Records	Files	
Name	Enabled	Tag Provider	Tag Path	Set	History Store	Sparkplug IDs
FF	true	FF	DeviceSet	Chariot SCADA Sparkplug	DeviceSet	<div><div>delete</div><div>edit</div></div>
MVI	true	MVI	RocketTray	Chariot SCADA Sparkplug	RocketTray	<div><div>delete</div><div>edit</div></div>

Figure 4.35: MQTT Transmitters

Publish

Subscribe

Sparkplug Explorer

Sparkplug Editor (beta)

Scripts

Broker Status

Log

▼ Sparkplug Messages

▼ spBv1.0

▶ USC_PLC_Server

▼ DeviceSet

▶ NDATA

▼ DDATA

▼ ConveyorBelt

Drives

▼ R03

MotionDevices

▼ R01

MotionDevices

▼ R02

MotionDevices

namespace spBv1.0

group_id DeviceSet

message_type DDATA

edge_node_id R01

device_id MotionDevices

```
{
  "timestamp": 1706554878075,
  "metrics": [
    {
      "name": "MotionDeviceR01/ParameterSet/JointAngleU",
      "timestamp": 1706554876651,
      "dataType": "Double",
      "value": 89.99854278564453
    },
    {
      "name": "MotionDeviceR01/ParameterSet/JointAngleR",
      "timestamp": 1706554876651,
      "dataType": "Double",
      "value": 0.0012352190678939223
    },
    {
      "name": "MotionDeviceR01/ParameterSet/JointAngleS",
      "timestamp": 1706554876653,
      "dataType": "Double",
      "value": 0.0018717440543696284
    },
    {
      "name": "MotionDeviceR01/ParameterSet/JointAngleL",
      "timestamp": 1706554876653,
      "dataType": "Double",
      "value": 7.486974936909974E-4
    },
    {
      "name": "MotionDeviceR01/ParameterSet/JointAngleB",
      "timestamp": 1706554876654,
      "dataType": "Double",
      "value": -90.00113677978516
    }
  ],
  "seq": 4
}
```

Figure 4.36: Subscribe to IG MQTT Server

An MQTT Transmitter was then configured for each of the AGs, as shown in Figure 4.35. The Tag Provider and the tag path for the transmitter are defined. These tags are then monitored for any value changes, upon which a new MQTT message with the updated value is published to the MQTT server. The tag values can then be modified from either writing a new value attribute to the server node via an OPC UA client connection or publishing a new message from a Sparkplug-Enabled MQTT client with the new value

metric to the MQTT server. Therefore, these transmitters serve to enable bidirectional data communication from either OPC UA or MQTT, and these value changes are reflected in both infrastructures i.e., the Ignition OPC UA Server and the Ignition MQTT Server. Figure 4.36 shows the subscription of the Sparkplug-Enabled MQTT client application MQTT.fx to the MQTT server on the IG.

4.3.3.3 MQTT ENGINE

The screenshot displays the 'Servers' tab in the MQTT Engine configuration. It features a 'Main' section with three fields: 'Name' (set to 'Chariot SCADA Sparkplug'), 'Enabled' (checked), and 'URL' (set to 'tcp://localhost:1883').

Main	
Name	Chariot SCADA Sparkplug <small>The friendly name of this MQTT Server Setting</small>
Enabled	<input checked="" type="checkbox"/> Enable this MQTT Server Setting
URL	tcp://localhost:1883 <small>The URL of the MQTT Server to connect to. Should be of the form tcp://mydomain.com:1883 or ssl://mydomain.com:8883</small>

Figure 4.37: Configure MQTT Server in MQTT Engine

In the MQTT Engine module, the Sparkplug B topic namespace is enabled by default. All MQTT topics published to the MQTT server will fall under the “spBv1.0” MQTT topic. This topic namespace indicates that the MQTT topic is using the Sparkplug B version 1.0 specification and also serves as an identifier for the protocol version and encoding used for the payload data. The “Server” tab in the MQTT Engine configuration provides a list of the MQTT Servers that the MQTT Engine module should connect to. A reference to the “Chariot SCADA Sparkplug” MQTT Server was configured in the MQTT Engine module and linked to the local TCP port (1883) of the MQTT Server from the MQTT Distributor module, shown in Figure 4.37. This allows for interaction with the MQTT data from SCADA applications in the Ignition platform.

4.3.3.3 CREATE GATEWAY NETWORK

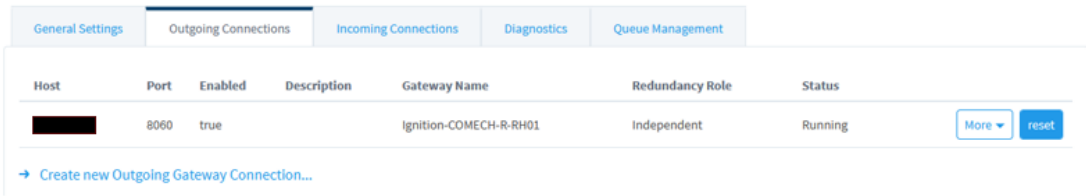


Figure 4.38: Configure Gateway Network Connection

TYPE	ACTION	TITLE
Server	refresh	[-] Ignition OPC UA Server
Object		[+] Devices
Object		[+] Server
Object		[-] Tag Providers
Object		[+] Ignition_teddyjetson_FF
Object		[+] Ignition_teddyjetson_MVI

Figure 4.39: Windows PC Ignition OPC UA Server

A gateway network was created on the IG, enabling data access to other approved IGs. The Ignition software was installed on a Windows PC in the manufacturing testbed, transforming it into an IG. An outgoing connection was then configured from the NVidia Jetson TX2 IG to the Windows PC IG, shown in Figure 4.38. On the Windows PC IG, a remote Tag Provider was created for each of the Tag Providers from the Jetson IG. In the OPC UA Server Settings, the “Expose Tag Providers” setting was once again enabled to bridge the Tag Providers to the local Ignition OPC UA Server. Figure 4.39 shows the local Ignition OPC UA Server, which includes the imported nodes from the Jetson Ignition OPC UA Server.

4.4 DISCUSSION

The diverse, widespread, and ever-changing characteristics of resources and devices, coupled with a vast array of data, pose significant challenges in the discovery,

access, processing, integration, and interpretation of real-world data in the IIoT. In the current state of the IIoT, intelligent manufacturing systems need to be able to access raw data from various resources over the network, which must be analyzed in order to extract knowledge.

This thesis focused on the development of a PnP framework that enables interoperable, scalable, and mobile communication between heterogeneous systems. The manufacturing use-case is realized through the development of AGs that collect data at the edge-of-the-network, and the development of an IG for the aggregation and distribution of process data throughout the network. The OPC UA framework was chosen for its information modeling features that enable data-rich and object-oriented representations for industrial systems. The MQTT protocol was chosen for its M2M communication capability in distributing information across the manufacturing ecosystem in a scalable manner. The Ignition platform was chosen due to its capability to easily aggregate heterogeneous data sources into a centralized data pipeline for the entire manufacturing environment using either OPC UA or MQTT, in addition to the F2F-enabling capabilities through the creation of gateway networks.

The framework utilizes these tools to enable the PnP characteristics for the IIoT architecture, transforming it into a PnP system. Interoperability within the PnP framework is enabled by OPC UA data modeling features and transportation mechanisms; these bidirectional communication capabilities facilitate data translation between the different systems, ensuring uniformity in data representation. Scalability is achieved through ACMs and centralized data aggregation in the Ignition platform, reducing manual effort for integrating new systems and increasing data availability for client applications. Mobility is

addressed by designing the PnP middleware to be easily transferable between systems, leveraging wireless networking and edge computing technologies. Overall, the PnP framework addresses key challenges in PnP applications, enabling interoperability, scalability, and mobility for intelligent manufacturing systems.

4.4.1 INTEROPERABILITY

The PnP framework serves to enable interoperability between heterogeneous industrial systems. Several aspects of the developed PnP framework contribute to this concept. This thesis explores the development of custom information models and the translation process between the OPC UA Framework and IIoT Protocols with the purpose of facilitating interoperability within the PnP framework. To enable interoperability in the PnP framework, the middleware translates data into a format that is understood by both the sending and receiving systems, which is the OPC UA framework. The communication between systems is bidirectional to enable cohesive interaction between operational equipment and IT applications. These effects are achieved through the design of custom information models using companion specifications to represent the inherent capabilities of the systems. Furthermore, the Ignition platform allows for bidirectional communication between the OPC UA and MQTT IIoT infrastructures, giving field devices more options to publish the data to the IG using either the OPC UA framework or the MQTT protocol. This ensures that there is uniformity of the manufacturing data between both the MQTT and OPC UA infrastructures.

4.4.2 SCALABILITY

The PnP framework serves to enable scalability for the manufacturing ecosystem. The key aspects of the PnP framework that contributes to the scalability concept are the

ability to reduce the manual configuration effort for integration of new systems, as well as to increase the availability of the manufacturing data within the industrial environment for client applications to access. The Ignition platform includes several driver modules that allow for seamless data access for specific types of devices. These drivers enable the writing of production data into Tag Providers without the need for intensive manual configuration. Furthermore, the bridging capabilities of the Ignition OPC UA Server are utilized to aggregate data sources into a centralized data pipeline to reduce the effort of data acquisition for intelligent manufacturing applications requiring data from heterogeneous data sources.

4.4.3 MOBILITY

The PnP framework was designed to be mobile in order to accommodate various trends of enabling technologies within the IIoT domain. These trends include the increasing utilization of wireless networking modalities for communication within industrial environments, as well as the use of edge computing for data acquisition and analysis at the edge of the network. These trends contribute towards the PnP concept in that PnP devices are meant to have the capability to be easily transferred between systems, and for the device to adjust dynamically to the new system without requiring intensive manual reconfiguration. By using wireless networking and edge computing, the PnP devices are able to be physically moved between systems more easily. Following these guidelines, the AGs and IG were developed on resource-constrained edge devices with a low physical footprint. The communication occurs wirelessly, meaning the PnP devices are not physically constrained to the location of the data source.

CHAPTER 5

CONCLUSION

5.1 SUMMARY OF WORK

The thesis introduces fundamental concepts such as the IoT, IIoT, and PnP. A thorough literature review on Plug and Produce within the IIoT is presented. This includes an examination of the general IIoT architecture and an exploration of IIoT connectivity. The taxonomy of Plug and Produce is defined in the context of the IoT and CPSs, drawing comparisons with related terms like Plug and Play and Plug and Work. It identifies and elaborates on the characteristics of Plug and Produce and discusses current research directions in key IIoT technologies enabling Plug and Produce. The review also delves into challenges associated with incorporating these technologies and proposes future research directions to address the identified gaps.

The thesis defines the use-case and requirements for the PnP framework in the Future Factories manufacturing testbed. The framework focuses on enabling PnP capabilities for automation equipment, robots, and machines. The capabilities refer to the defined PnP characteristics, which are interoperability, scalability, and mobility of the system. The framework also aims to facilitate horizontal and vertical communication between all hardware and software entities to foster the innovation of new business

functions. Currently, the Future Factories manufacturing testbed does not implement the use of an MES system; however, the architecture and various system components of the PnP framework were carefully selected in order to allow for seamless interactivity with such systems upon their eventual introduction into the testbed. The thesis then provides an overview of the testbed's components, emphasizing the capabilities of OPC UA in enabling interoperability for manufacturing systems. The interoperability-enabling properties of the OPC UA framework are detailed, including features like information modeling and companion specifications. Additionally, the process of developing custom information models for definition of asset capabilities in the manufacturing environment is explained. The role of various modules and software integrations in building the IG platform as a scalability solution for PnP systems is discussed, followed by an explanation of the development methodology for the PnP framework. Lastly, the thesis presents a roadmap for the integration of the PnP framework and associated IIoT technologies in brownfield manufacturing scenarios.

The thesis showcases the developed architecture of the PnP framework for the Future Factories manufacturing testbed. This is followed by outlining the development approach for the AGs for the manufacturing processes by employing custom information models for specific stations within the manufacturing testbed. The thesis then details the configuration process for IGs to facilitate scalability across the span of the IIoT architecture of the manufacturing system. The system's communication protocol agnostic nature is emphasized, achieved through a combination of software packages and libraries.

5.2 FUTURE WORK

There are additional opportunities to enhance the capabilities of the PnP framework for the Future Factories manufacturing testbed. One limitation of the PnP framework is the lack of utilization of ACMs for reducing the effort required for device integration. Devices that have compatible drivers within the Ignition platform or that feature inherent OPC UA or MQTT capabilities can be seamlessly connected to the Ignition Gateway. However, for devices that don't meet these requirements, such as legacy industrial machine systems, the process of building information models, acquiring data, and populating the OPC UA server is much too variable and manually intensive. For future work, the PnP framework should incorporate various types of ACMs to reduce the manual configuration effort for system integration. The following ACMs would serve to achieve this outcome:

Table 5.1: ACMs for PnP Framework in Future Work

Device Discovery	Allows for the OPC UA server to scan for available devices on the network using the TCP/IP protocol to identify potential new device candidates to collect information from.
Dynamic Information Exchange	Send queries to browse the address spaces of discovered devices on the network to obtain information about their process parameters. When new parameters are defined in the industrial system, the associated changes to the information model should happen dynamically during server runtime.
Automatic Information Modeling	Based on the received process parameter information, the information model for the device would be automatically generated. However, this process would require a minimum level of manual configuration, as parameters are not always able to fully describe the characteristics of a system e.g., the "HasComponent" references to define the relationship between physical components within the industrial system.

Another limitation of the PnP framework is the inability to represent all of the attributes of an OPC UA node within the MQTT infrastructure. For example, the MQTT message payload data contains the name, timestamp, datatype, and value metrics. Notably missing from this payload structure is the Object/Variable/Data Type, the “BrowseName” and “DisplayName” attributes, references and reference types, etc. Devices that are subscribed to the MQTT server of the IG lose a portion of the metadata that serves to describe the characteristics of the OPC UA node. This challenge can be mitigated by defining a custom MQTT payload structure for MQTT messages to include the definition of these OPC UA attributes. However, this solution faces potential compatibility issues with devices that publish MQTT messages with a fixed payload structure. Solving this issue would require developing an automated mechanism that subscribes to the publishing device from an MQTT client, restructures the message payload to include the additional OPC UA attributes, and republishes the reformatted MQTT message to the IG.

5.3 SITUATION OF RESEARCH

This investigation into PnP connectivity for manufacturing systems aims to extend the research conducted in the Smart Manufacturing domain in the Future Factories laboratory at the University of South Carolina’s McNair Center. Analyses on Smart Manufacturing were conducted, such as identifying major manufacturing paradigms for factories of the future [126] and investigating optimization of Smart Manufacturing systems from early-state analyses of smart products during the beginning-of-life (BOL) stages [127]. Other studies aimed to identify mechanical attributes [128] and perform semantic segmentation [129] for enhanced event comprehension in the manufacturing environment. The mechanical attributes are semantically represented within the PnP

framework using OPC UA information modeling to define the characteristics of the industrial assets, and the system framework provides the capability to define manufacturing events through conditional requirements. Earlier projects also analyzed cognition in DTs [130], integrated reinforcement learning with DTs to enhance intelligence [131], explored multi-modal robotic health in the IoT [132], developed a mapping method for product-centered DTs to automatically analyze product manufacturability [133], created a DT for an Automated Fiber Placement (AFP) manufacturing system [134], and delved into intelligent virtual commissioning [135]. The PnP framework enables communication via the common OPC UA framework to the Siemens *Process Simulate* software used for virtual commissioning, digital twinning, and process simulation. Other notable works also include enabling motion-capture-based calibration for industrial robots [136] and integration of semantic web technologies to for fault-tolerant autonomous manufacturing [137]. An analog and a multi-modal time-series dataset were generated from the rocket assembly process in the Future Factories manufacturing testbed and were made publicly-available to promote further development of Smart Manufacturing applications in real-world industrial environments [138]. The common objective of these research works is to enable the autonomy of manufacturing processes by incorporating Artificial Intelligence (AI) into manufacturing through diverse methodologies, and the PnP framework aims to further enhance the collaboration of these systems by facilitating seamless and interactive communication between the various hardware and software components. Additionally, the current work builds upon the preliminary design efforts directed at establishing the Future Factories testbed [139].

REFERENCES

- [1] A. M. Houyou and H.-P. Huth, “Internet of Things at Work: Enabling Plug-and-Work in Automation Networks,” 2011, Accessed: Nov. 16, 2023. [Online]. Available: <https://www.researchgate.net/publication/230577679>
- [2] “Internet of Things [IoT] Market Size, Share & Growth by 2030.” Accessed: Feb. 06, 2024. [Online]. Available: <https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307>
- [3] M. Suárez-Albela, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, “A Practical Evaluation of a High-Security Energy-Efficient Gateway for IoT Fog Computing Applications,” *Sensors* 2017, Vol. 17, Page 1978, vol. 17, no. 9, p. 1978, Aug. 2017, doi: 10.3390/S17091978.
- [4] J. Robert, S. Kubler, N. Kolbe, A. Cerioni, E. Gastaud, and K. Främling, “Open IoT Ecosystem for Enhanced Interoperability in Smart Cities—Example of Métropole De Lyon,” *Sensors* 2017, Vol. 17, Page 2849, vol. 17, no. 12, p. 2849, Dec. 2017, doi: 10.3390/S17122849.
- [5] T. M. Fernández-Caramés, “An intelligent power outlet system for the smart home of the internet of things,” *Int J Distrib Sens Netw*, vol. 2015, 2015, doi: 10.1155/2015/214805.
- [6] W. Mao, Z. Zhao, Z. Chang, G. Min, and W. Gao, “Energy-Efficient Industrial Internet of Things: Overview and Open Issues,” *IEEE Trans Industr Inform*, vol. 17, no. 11, pp. 7225–7237, Nov. 2021, doi: 10.1109/TII.2021.3067026.
- [7] J. J. P. C. Rodrigues *et al.*, “Enabling Technologies for the Internet of Health Things,” *IEEE Access*, vol. 6, pp. 13129–13141, Jan. 2018, doi: 10.1109/ACCESS.2017.2789329.

- [8] P. Fraga-Lamas, T. M. Fernández-Caramés, and L. Castedo, "Towards the Internet of Smart Trains: A Review on Industrial IoT-Connected Railways," *Sensors (Basel)*, vol. 17, no. 6, Jun. 2017, doi: 10.3390/S17061457.
- [9] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris, "A Review of Machine Learning and IoT in Smart Transportation," *Future Internet 2019, Vol. 11, Page 94*, vol. 11, no. 4, p. 94, Apr. 2019, doi: 10.3390/FI11040094.
- [10] A. Karmakar, N. Dey, T. Baral, M. Chowdhury, and M. Rehan, "Industrial internet of things: A review," *2019 International Conference on Opto-Electronics and Applied Optics, Optronix 2019*, Mar. 2019, doi: 10.1109/OPTRONIX.2019.8862436.
- [11] F. Tao, Q. Qi, A. Liu, and A. Kusiak, "Data-driven smart manufacturing," *J Manuf Syst*, vol. 48, pp. 157–169, Jul. 2018, doi: 10.1016/J.JMSY.2018.01.006.
- [12] D. Mourtzis, E. Vlachou, and N. Milas, "Industrial Big Data as a Result of IoT Adoption in Manufacturing," *Procedia CIRP*, vol. 55, pp. 290–295, Jan. 2016, doi: 10.1016/J.PROCIR.2016.07.038.
- [13] J. Lee, E. Lapira, B. Bagheri, and H. an Kao, "Recent advances and trends in predictive manufacturing systems in big data environment," *Manuf Lett*, vol. 1, no. 1, pp. 38–41, Oct. 2013, doi: 10.1016/J.MFGLET.2013.09.005.
- [14] J. Lenz, T. Wuest, and E. Westkämper, "Holistic approach to machine tool data analytics," *J Manuf Syst*, vol. 48, pp. 180–191, Jul. 2018, doi: 10.1016/J.JMSY.2018.03.003.
- [15] N. Mircica, "Cyber-Physical Systems for Cognitive Industrial Internet of Things: Sensory Big Data, Smart Mobile Devices, and Automated Manufacturing Processes.," *Analysis and Metaphysics*, vol. 18, pp. 37–44, Jan. 2019, Accessed: Oct. 17, 2022. [Online]. Available: <https://go.gale.com/ps/i.do?p=AONE&sw=w&issn=15848574&v=2.1&it=r&id=GALE%7CA611679250&sid=googleScholar&linkaccess=fulltext>
- [16] Z. Zhao *et al.*, "A Novel Framework of Three-Hierarchical Offloading Optimization for MEC in Industrial IoT Networks," *IEEE Trans Industr Inform*, vol. 16, no. 8, pp. 5424–5434, Aug. 2020, doi: 10.1109/TII.2019.2949348.
- [17] A. Sinha, E. Bernardes, and R. Calderon, "Digital supply networks : transform your supply chain and gain competitive advantage with disruptive technology and reimagined processes," pp. 49–60, 2020, Accessed: Nov.

- 30, 2023. [Online]. Available:
https://www.researchgate.net/publication/342530439_Digital_Supply_Networks_Transform_Your_Supply_Chain_and_Gain_Competitive_Advantage_with_Disruptive_Technology_and_Reimagined_Processes
- [18] M. Hankel, B. R.- Zvei, and undefined 2015, “The reference architectural model industrie 4.0 (rami 4.0),” *zvei.org*, Accessed: Feb. 05, 2024. [Online]. Available:
https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2015/april/Das_Referenzarchitekturmodell_Industrie_4.0__RAMI_4.0_/ZVEI-Industrie-40-RAMI-40-English.pdf
 - [19] H. D. Nguyen, K. P. Tran, X. Zeng, L. Koehl, P. Castagliola, and P. Bruniaux, “Industrial Internet of Things, Big Data, and Artificial Intelligence in the Smart Factory: a survey and perspective,” pp. 72–76, Jul. 2019, Accessed: Nov. 15, 2023. [Online]. Available: <https://hal.science/hal-02268119>
 - [20] X. Ye, J. Jiang, C. Lee, N. Kim, M. Yu, and S. H. Hong, “Toward the Plug-and-Produce Capability for Industry 4.0: An Asset Administration Shell Approach,” *IEEE Industrial Electronics Magazine*, vol. 14, no. 4, pp. 146–157, Dec. 2020, doi: 10.1109/MIE.2020.3010492.
 - [21] M. Wollschlaeger, R. Schrieber, and M. Ullemeyer, “Life-cycle-management in automation-models and strategies,” *IECON Proceedings (Industrial Electronics Conference)*, pp. 2578–2584, Feb. 2014, doi: 10.1109/IECON.2014.7048869.
 - [22] Ł. Hady, M. Dyląg, and G. Wozny, “Investment cost estimation and calculation of chemical plants with classical and modular approaches,” *Chemical and Process Engineering*, vol. Vol. 30, z. 2, pp. 319–340, 2009.
 - [23] C. Bramsiepe *et al.*, “Low-cost small scale processing technologies for production applications in various environments—Mass produced factories,” *Chemical Engineering and Processing: Process Intensification*, vol. 51, pp. 32–52, Jan. 2012, doi: 10.1016/J.CEP.2011.08.005.
 - [24] K. Dorofeev, C. H. Cheng, M. Guedes, P. Ferreira, S. Profanter, and A. Zoitl, “Device adapter concept towards enabling plug&produce production environments,” *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pp. 1–8, Jun. 2017, doi: 10.1109/ETFA.2017.8247570.

- [25] L. Duerkop, H. Trsek, J. Jasperneite, and L. Wisniewski, "Towards autoconfiguration of industrial automation systems: A case study using Profinet IO," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2012, doi: 10.1109/ETFA.2012.6489654.
- [26] H. Koziol, A. Burger, M. Platenius-Mohr, J. Ruckert, and G. Stomberg, "OpenPnP: A Plug-And-Produce Architecture for the Industrial Internet of Things," *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*, pp. 131–140, May 2019, doi: 10.1109/ICSE-SEIP.2019.00022.
- [27] D. Garlan, "Software architecture: A travelogue," *Future of Software Engineering, FOSE 2014 - Proceedings*, pp. 29–39, May 2014, doi: 10.1145/2593882.2593886.
- [28] P. Danny, P. Ferreira, N. Lohse, and M. Guedes, "An AutomationML model for plug-and-produce assembly systems," *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, pp. 849–854, Nov. 2017, doi: 10.1109/INDIN.2017.8104883.
- [29] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Trans Industr Inform*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014, doi: 10.1109/TII.2014.2300753.
- [30] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, Oct. 2015, doi: 10.1109/COMST.2015.2444095.
- [31] Y. Liao, E. D. F. R. Loures, and F. Deschamps, "Industrial Internet of Things: A Systematic Literature Review and Insights," *IEEE Internet Things J*, vol. 5, no. 6, pp. 4515–4525, Dec. 2018, doi: 10.1109/JIOT.2018.2834151.
- [32] P. Zheng *et al.*, "Smart manufacturing systems for Industry 4.0: Conceptual framework, scenarios, and future perspectives," *Frontiers of Mechanical Engineering*, vol. 13, no. 2, pp. 137–150, Jun. 2018, doi: 10.1007/S11465-018-0499-5/METRICS.
- [33] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial internet of things: Challenges, opportunities, and directions," *IEEE Trans Industr Inform*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018, doi: 10.1109/TII.2018.2852491.

- [34] P. K. Malik *et al.*, “Industrial Internet of Things and its Applications in Industry 4.0: State of The Art,” *Comput Commun*, vol. 166, pp. 125–139, Jan. 2021, doi: 10.1016/J.COMCOM.2020.11.016.
- [35] P. Patel, M. I. Ali, and A. Sheth, “From Raw Data to Smart Manufacturing: AI and Semantic Web of Things for Industry 4.0,” *IEEE Intell Syst*, vol. 33, pp. 79–86, Jul. 2018, doi: 10.1109/MIS.2018.043741325.
- [36] Ó. Blanco-Novoa, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, “An Electricity Price-Aware Open-Source Smart Socket for the Internet of Energy,” *Sensors 2017, Vol. 17, Page 643*, vol. 17, no. 3, p. 643, Mar. 2017, doi: 10.3390/S17030643.
- [37] M. Weyrich and C. Ebert, “Reference architectures for the internet of things,” *IEEE Softw*, vol. 33, no. 1, pp. 112–116, Jan. 2016, doi: 10.1109/MS.2016.20.
- [38] “The Industrial Internet of Things Connectivity Framework An Industry IoT Consortium Foundational Document”, Accessed: Nov. 27, 2023. [Online]. Available: <https://www.iiconsortium.org/iicf/>
- [39] “The Industrial Internet Reference Architecture - Industry IoT Consortium.” Accessed: Feb. 05, 2024. [Online]. Available: <https://www.iiconsortium.org/iira/>
- [40] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, “The industrial internet of things (IIoT): An analysis framework,” *Comput Ind*, vol. 101, pp. 1–12, Oct. 2018, doi: 10.1016/J.COMPIND.2018.04.015.
- [41] Y. Ren, R. Xie, F. R. Yu, T. Huang, and Y. Liu, “Potential Identity Resolution Systems for the Industrial Internet of Things: A Survey,” *IEEE Communications Surveys and Tutorials*, vol. 23, no. 1, pp. 391–430, Jan. 2021, doi: 10.1109/COMST.2020.3045136.
- [42] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, “Toward better horizontal integration among IoT services,” *IEEE Communications Magazine*, vol. 53, no. 9, pp. 72–79, Sep. 2015, doi: 10.1109/MCOM.2015.7263375.
- [43] J. Kiljander *et al.*, “Semantic interoperability architecture for pervasive computing and internet of things,” *IEEE Access*, vol. 2, pp. 856–873, 2014, doi: 10.1109/ACCESS.2014.2347992.

- [44] A. Carvalho, N. O' Mahony, L. Krpalkova, S. Campbell, J. Walsh, and P. Doody, "Edge Computing Applied to Industrial Machines," *Procedia Manuf*, vol. 38, pp. 178–185, Jan. 2019, doi: 10.1016/J.PROMFG.2020.01.024.
- [45] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang, "Edge Computing in IoT-Based Manufacturing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 103–109, 2018, doi: 10.1109/MCOM.2018.1701231.
- [46] G. Pedone and I. Mezgár, "Model similarity evidence and interoperability affinity in cloud-ready Industry 4.0 technologies," *Comput Ind*, vol. 100, pp. 278–286, Sep. 2018, doi: 10.1016/J.COMPIND.2018.05.003.
- [47] C. Liu, Z. Su, X. Xu, and Y. Lu, "Service-oriented industrial internet of things gateway for cloud manufacturing," *Robot Comput Integr Manuf*, vol. 73, p. 102217, Feb. 2022, doi: 10.1016/J.RCIM.2021.102217.
- [48] P. K. Illa and N. Padhi, "Practical Guide to Smart Factory Transition Using IoT, Big Data and Edge Analytics," *IEEE Access*, vol. 6, pp. 55162–55170, 2018, doi: 10.1109/ACCESS.2018.2872799.
- [49] O. Throne and G. Lazaroiu, "Internet of Things-enabled Sustainability, Industrial Big Data Analytics, and Deep Learning-assisted Smart Process Planning in Cyber-Physical Manufacturing Systems.," *Economics, Management, and Financial Markets*, vol. 15, no. 4, pp. 49–59, Dec. 2020, Accessed: Oct. 17, 2022. [Online]. Available: <https://go.gale.com/ps/i.do?p=AONE&sw=w&issn=18423191&v=2.1&it=r&id=GALE%7CA647255686&sid=googleScholar&linkaccess=fulltext>
- [50] Y. Lu, "Industry 4.0: A survey on technologies, applications and open research issues," *J Ind Inf Integr*, vol. 6, pp. 1–10, Jun. 2017, doi: 10.1016/J.JII.2017.04.005.
- [51] P. Boobalan *et al.*, "Fusion of Federated Learning and Industrial Internet of Things: A survey," *Computer Networks*, vol. 212, p. 109048, Jul. 2022, doi: 10.1016/J.COMNET.2022.109048.
- [52] J. Schmitt, J. Bönig, T. Borggräfe, G. Beiting, and J. Deuse, "Predictive model-based quality inspection using Machine Learning and Edge Cloud Computing," *Advanced Engineering Informatics*, vol. 45, p. 101101, Aug. 2020, doi: 10.1016/J.AEI.2020.101101.

- [53] “Plug and play Definition & Meaning - Merriam-Webster.” Accessed: Feb. 05, 2024. [Online]. Available: <https://www.merriam-webster.com/dictionary/plug%20and%20play>
- [54] S. Liu, J. A. Guzzo, L. Zhang, D. W. Smith, J. Lazos, and M. Grossner, “Plug-and-play sensor platform for legacy industrial machine monitoring,” *International Symposium on Flexible Automation, ISFA 2016*, pp. 432–435, Dec. 2016, doi: 10.1109/ISFA.2016.7790202.
- [55] A. Ghazanfari, M. Hamzeh, and Y. A. R. I. Mohamed, “A Resilient Plug-and-Play Decentralized Control for DC Parking Lots,” *IEEE Trans Smart Grid*, vol. 9, no. 3, pp. 1930–1942, May 2018, doi: 10.1109/TSG.2016.2602759.
- [56] B. Bordel, D. S. De Rivera, and R. Alcarria, “Plug-and-play transducers in cyber-physical systems for device-driven applications,” *Proceedings - 2016 10th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2016*, pp. 316–321, Dec. 2016, doi: 10.1109/IMIS.2016.68.
- [57] N. Matthys *et al.*, “ μ pnP-Mesh: The plug-and-play mesh network for the Internet of Things,” *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, pp. 311–315, 2015, doi: 10.1109/WF-IOT.2015.7389072.
- [58] D. L. Hernández-Rojas, T. M. Fernández-Caramés, P. Fraga-Lamas, and C. J. Escudero, “A Plug-and-Play Human-Centered Virtual TEDS Architecture for the Web of Things,” *Sensors 2018, Vol. 18, Page 2052*, vol. 18, no. 7, p. 2052, Jun. 2018, doi: 10.3390/S18072052.
- [59] W. Kim, H. Ko, H. Yun, J. Sung, S. Kim, and J. Nam, “A generic Internet of things (IoT) platform supporting plug-and-play device management based on the semantic web,” *J Ambient Intell Humaniz Comput*, vol. 1, pp. 1–11, Sep. 2019, doi: 10.1007/S12652-019-01464-2/TABLES/1.
- [60] M. Schleipen, E. Selyansky, R. Henssen, and T. Bischoff, “Multi-level user and role concept for a secure plug-and-work based on OPC UA and AutomationML,” *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2015-October, Oct. 2015, doi: 10.1109/ETFA.2015.7301640.
- [61] M. Schleipen, A. Lüder, O. Sauer, H. Flatt, and J. Jasperneite, “Requirements and concept for Plug-and-Work: Adaptivity in the context of

- Industry 4.0,” *At-Automatisierungstechnik*, vol. 63, no. 10, pp. 801–820, Oct. 2015, doi: 10.1515/AUTO-2015-0015.
- [62] J. M. Gutierrez-Guerrero and J. A. Holgado-Terriza, “Automatic Configuration of OPC UA for Industrial Internet of Things Environments,” *Electronics* 2019, Vol. 8, Page 600, vol. 8, no. 6, p. 600, May 2019, doi: 10.3390/ELECTRONICS8060600.
 - [63] B. Madiwalar, B. Schneider, and S. Profanter, “Plug and Produce for Industry 4.0 using Software-defined Networking and OPC UA,” *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2019-September, pp. 126–133, Sep. 2019, doi: 10.1109/ETFA.2019.8869525.
 - [64] A. Hazra, M. Adhikari, T. Amgoth, and S. N. Srirama, “A Comprehensive Survey on Interoperability for IIoT: Taxonomy, Standards, and Future Directions,” *ACM Computing Surveys (CSUR)*, vol. 55, no. 1, Nov. 2021, doi: 10.1145/3485130.
 - [65] V. R. Konduru and M. R. Bharamagoudra, “Challenges and solutions of interoperability on IoT: How far have we come in resolving the IoT interoperability issues,” *Proceedings of the 2017 International Conference On Smart Technology for Smart Nation, SmartTechCon 2017*, pp. 572–576, May 2018, doi: 10.1109/SMARTTECHCON.2017.8358436.
 - [66] H. Derhamy, J. Eliasson, and J. Delsing, “IoT Interoperability - On-Demand and Low Latency Transparent Multiprotocol Translator,” *IEEE Internet Things J*, vol. 4, no. 5, pp. 1754–1763, Oct. 2017, doi: 10.1109/JIOT.2017.2697718.
 - [67] S. Javed, M. Usman, F. Sandin, M. Liwicki, and H. Mokayed, “Deep Ontology Alignment Using a Natural Language Processing Approach for Automatic M2M Translation in IIoT,” *Sensors* 2023, Vol. 23, Page 8427, vol. 23, no. 20, p. 8427, Oct. 2023, doi: 10.3390/S23208427.
 - [68] R. Schiekofer, S. Grimm, M. M. Brandt, and M. Weyrich, “A formal mapping between OPC UA and the semantic web,” *IEEE International Conference on Industrial Informatics (INDIN)*, vol. 2019-July, pp. 33–40, Jul. 2019, doi: 10.1109/INDIN41052.2019.8972102.
 - [69] R. Alt, P. Wintersohle, H. Schweizer, M. Wollschlaeger, and K. Schmitz, “Interoperable information model of a pneumatic handling system for plug-and-produce,” pp. 19–26, Nov. 2020, doi: 10.25368/2020.65.

- [70] J. Kim *et al.*, “Standard-based IoT platforms interworking: Implementation, experiences, and lessons learned,” *IEEE Communications Magazine*, vol. 54, no. 7, pp. 48–54, Jul. 2016, doi: 10.1109/MCOM.2016.7514163.
- [71] G. Aceto, V. Persico, and A. Pescapé, “A Survey on Information and Communication Technologies for Industry 4.0: State-of-the-Art, Taxonomies, Perspectives, and Challenges,” *IEEE Communications Surveys and Tutorials*, vol. 21, no. 4, pp. 3467–3501, Oct. 2019, doi: 10.1109/COMST.2019.2938259.
- [72] “UA Part 6: Mappings - 7.4 OPC UA HTTPS.” Accessed: Dec. 10, 2023. [Online]. Available: <https://reference.opcfoundation.org/Core/Part6/v105/docs/7.4>
- [73] “UA Part 4: Services - 4.1 Service Set model.” Accessed: Feb. 05, 2024. [Online]. Available: <https://reference.opcfoundation.org/Core/Part4/v104/docs/4.1>
- [74] P. Ferreira, “AN AGENT-BASED SELF-CONFIGURATION METHODOLOGY FOR MODULAR ASSEMBLY SYSTEMS,” Doctoral Dissertation, University of Nottingham, 2011. Accessed: Nov. 29, 2023. [Online]. Available: https://eprints.nottingham.ac.uk/12325/1/Thesis_Full_Version.pdf
- [75] P. Desai, A. Sheth, and P. Anantharam, “Semantic Gateway as a Service Architecture for IoT Interoperability,” *Proceedings - 2015 IEEE 3rd International Conference on Mobile Services, MS 2015*, pp. 313–319, Aug. 2015, doi: 10.1109/MOBSERV.2015.51.
- [76] H. Derhamy, J. Ronnholm, J. Delsing, J. Eliasson, and J. Van Deventer, “Protocol interoperability of OPC UA in service oriented architectures,” *Proceedings - 2017 IEEE 15th International Conference on Industrial Informatics, INDIN 2017*, pp. 44–50, Nov. 2017, doi: 10.1109/INDIN.2017.8104744.
- [77] B. Katti, C. Plociennik, M. Ruskowski, and M. Schweitzer, “SA-OPC-UA: Introducing Semantics to OPC-UA Application Methods,” *IEEE International Conference on Automation Science and Engineering*, vol. 2018-August, pp. 1189–1196, Dec. 2018, doi: 10.1109/COASE.2018.8560467.
- [78] F. Pauker, S. Wolny, S. M. Fallah, and M. Wimmer, “UML2OPC-UA Transforming UML Class Diagrams to OPC UA Information Models,”

- Procedia CIRP*, vol. 67, pp. 128–133, Jan. 2018, doi: 10.1016/J.PROCIR.2017.12.188.
- [79] J. Koo, S. R. Oh, and Y. G. Kim, “Device Identification Interoperability in Heterogeneous IoT Platforms †,” *undefined*, vol. 19, no. 6, pp. 6–8, Mar. 2019, doi: 10.3390/S19061433.
 - [80] S. Cavalieri and F. Chiacchio, “Analysis of OPC UA performances,” *Comput Stand Interfaces*, vol. 36, no. 1, pp. 165–177, Nov. 2013, doi: 10.1016/J.CSI.2013.06.004.
 - [81] R. Alt, J. Malzahn, H. Murrenhoff, and K. Schmitz, “A Survey of Industrial Internet of Things in the Field of Fluid Power: Basic Concept and Requirements for Plug-and-Produce,” *BATH/ASME 2018 Symposium on Fluid Power and Motion Control, FPMC 2018*, Nov. 2018, doi: 10.1115/FPMC2018-8833.
 - [82] Z. Liu and P. Bellot, “A configuration tool for MQTT based OPC UA PubSub,” *Proceedings - 2020 RIVF International Conference on Computing and Communication Technologies, RIVF 2020*, Oct. 2020, doi: 10.1109/RIVF48685.2020.9140792.
 - [83] P. Drahos, E. Kucera, O. Haffner, and I. Klimo, “Trends in industrial communication and OPC UA,” *Proceedings of the 29th International Conference on Cybernetics and Informatics, K and I 2018*, vol. 2018-January, pp. 1–5, Apr. 2018, doi: 10.1109/CYBERI.2018.8337560.
 - [84] A. Burger, H. Koziolk, J. Rückert, M. Platenius-Mohr, and G. Stomberg, “Bottleneck Identification and Performance Mod-eling of OPC UA Communication Models,” *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pp. 12–19, 2019, doi: 10.1145/3297663.
 - [85] A. Farahzadi, P. Shams, J. Rezazadeh, and R. Farahbakhsh, “Middleware technologies for cloud of things: a survey,” *Digital Communications and Networks*, vol. 4, no. 3, pp. 176–188, Aug. 2018, doi: 10.1016/J.DCAN.2017.04.005.
 - [86] G. Jo, S. H. Jang, and J. Jeong, “Design and Implementation of CPPS and Edge Computing Architecture based on OPC UA Server,” *Procedia Comput Sci*, vol. 155, pp. 97–104, Jan. 2019, doi: 10.1016/J.PROCS.2019.08.017.

- [87] L. Beňo, R. Pribiš, and R. Leskovský, "Processing data from OPC UA server by using Edge and Cloud computing," *IFAC-PapersOnLine*, vol. 52, no. 27, pp. 240–245, Jan. 2019, doi: 10.1016/J.IFACOL.2019.12.645.
- [88] T. Mai, H. Yao, S. Guo, and Y. Liu, "In-Network Computing Powered Mobile Edge: Toward High Performance Industrial IoT," *IEEE Netw*, vol. 35, no. 1, pp. 289–295, Mar. 2021, doi: 10.1109/MNET.021.2000318.
- [89] S. Singh, "Optimize cloud computations using edge computing," *2017 International Conference on Big Data, IoT and Data Science, BID 2017*, vol. 2018-January, pp. 49–53, Apr. 2018, doi: 10.1109/BID.2017.8336572.
- [90] A. Alnoman, S. K. Sharma, W. Ejaz, and A. Anpalagan, "Emerging edge computing technologies for distributed IoT systems," *IEEE Netw*, vol. 33, no. 6, pp. 140–147, Nov. 2019, doi: 10.1109/MNET.2019.1800543.
- [91] T. Mai, H. Yao, S. Guo, and Y. Liu, "In-Network Computing Powered Mobile Edge: Toward High Performance Industrial IoT," *IEEE Netw*, vol. 35, no. 1, pp. 289–295, Mar. 2021, doi: 10.1109/MNET.021.2000318.
- [92] P. Desai, A. Sheth, and P. Anantharam, "Semantic Gateway as a Service architecture for IoT Interoperability."
- [93] A. Bhattacharya and P. De, "A survey of adaptation techniques in computation offloading," *Journal of Network and Computer Applications*, vol. 78, pp. 97–115, Jan. 2017, doi: 10.1016/J.JNCA.2016.10.023.
- [94] H. Guo, J. Liu, and J. Lv, "Toward Intelligent Task Offloading at the Edge," *IEEE Netw*, vol. 34, no. 2, pp. 128–134, Mar. 2020, doi: 10.1109/MNET.001.1900200.
- [95] S. Aljanabi and A. Chalechale, "Improving IoT Services Using a Hybrid Fog-Cloud Offloading," *IEEE Access*, vol. 9, pp. 13775–13788, 2021, doi: 10.1109/ACCESS.2021.3052458.
- [96] W. Sun, J. Liu, and Y. Yue, "AI-enhanced offloading in edge computing: When machine learning meets industrial IoT," *IEEE Netw*, vol. 33, no. 5, pp. 68–74, Sep. 2019, doi: 10.1109/MNET.001.1800510.
- [97] A. Ghosh, O. Khalid, R. N. B. Rais, A. Rehman, S. U. R. Malik, and I. A. Khan, "Data offloading in IoT environments: modeling, analysis, and verification," *EURASIP J Wirel Commun Netw*, vol. 2019, no. 1, pp. 1–23, Dec. 2019, doi: 10.1186/S13638-019-1358-8/FIGURES/14.

- [98] M. Aazam, S. Zeadally, and K. A. Harras, “Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities,” *Future Generation Computer Systems*, vol. 87, pp. 278–289, Oct. 2018, doi: 10.1016/J.FUTURE.2018.04.057.
- [99] R. Alt and K. Schmitz, “Basic requirements for Plug-and-Produce of 14.0 fluid power systems,” *Proceedings of the 8th International Conference on Fluid Power and Mechatronics, FPM 2019*, pp. 1440–1448, Apr. 2019, doi: 10.1109/FPM45753.2019.9035813.
- [100] A. Giret, E. Garcia, and V. Botti, “An engineering framework for Service-Oriented Intelligent Manufacturing Systems,” *Comput Ind*, vol. 81, pp. 116–127, Sep. 2016, doi: 10.1016/J.COMPIND.2016.02.002.
- [101] J. Imtiaz, J. Jasperneite, K. Weber, F. J. Goetz, and G. Lessmann, “A novel method for Auto configuration of Realtime Ethernet Networks,” *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pp. 861–868, 2008, doi: 10.1109/ETFA.2008.4638498.
- [102] L. Dürkop, H. Trsek, J. Otto, and J. Jasperneite, “A field level architecture for reconfigurable real-time automation systems,” *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, 2014, doi: 10.1109/WFCS.2014.6837601.
- [103] M. Saqlain, M. Piao, Y. Shim, and J. Y. Lee, “Framework of an IoT-based Industrial Data Management for Smart Manufacturing,” *Journal of Sensor and Actuator Networks 2019, Vol. 8, Page 25*, vol. 8, no. 2, p. 25, Apr. 2019, doi: 10.3390/JSAN8020025.
- [104] M. H. Farzaneh and A. Knoll, “An ontology-based Plug-and-Play approach for in-vehicle Time-Sensitive Networking (TSN),” *IEEE Annual Information Technology, Electronics and Mobile Communication Conference*, Nov. 2016, doi: 10.1109/IEMCON.2016.7746299.
- [105] R. Kiesel, K. Stichling, P. Hemmers, T. Vollmer, and R. H. Schmitt, “Quantification of Influence of 5G Technology Implementation on Process Performance in Production,” *Procedia CIRP*, vol. 104, pp. 104–109, Jan. 2021, doi: 10.1016/J.PROCIR.2021.11.018.
- [106] M. Barring *et al.*, “5G Enabled Manufacturing Evaluation for Data-Driven Decision-Making,” *Procedia CIRP*, vol. 72, pp. 266–271, Jan. 2018, doi: 10.1016/J.PROCIR.2018.03.169.

- [107] J. S. Walia, H. Hämmäinen, K. Kilkki, and S. Yrjölä, “5G network slicing strategies for a smart factory,” *Comput Ind*, vol. 111, pp. 108–120, Oct. 2019, doi: 10.1016/J.COMPIND.2019.07.006.
- [108] M. Gidlund, T. Lennvall, and J. Akerberg, “Will 5G become yet another wireless technology for industrial automation?,” *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 1319–1324, Apr. 2017, doi: 10.1109/ICIT.2017.7915554.
- [109] J. Åkerberg, M. Gidlund, and M. Bjorkman, “Future research challenges in wireless sensor and actuator networks targeting industrial automation,” *IEEE International Conference on Industrial Informatics (INDIN)*, pp. 410–415, 2011, doi: 10.1109/INDIN.2011.6034912.
- [110] “Programming an OPC UA .NET Client with C# for the SIMATIC NET OPC UA Server”, Accessed: Dec. 03, 2023. [Online]. Available: <https://support.industry.siemens.com/cs/ww/en/view/42014088>
- [111] “UA Companion Specifications - OPC Foundation.” Accessed: Dec. 03, 2023. [Online]. Available: <https://opcfoundation.org/about/opc-technologies/opc-ua/ua-companion-specifications/>
- [112] “Cloud Library - Overall Architecture and Use Cases - OPC Foundation.” Accessed: Dec. 03, 2023. [Online]. Available: <https://opcfoundation.org/developer-tools/documents/view/201>
- [113] “Devices - OPC Foundation.” Accessed: Dec. 03, 2023. [Online]. Available: <https://opcfoundation.org/developer-tools/documents/view/197>
- [114] “How to create custom OPC UA Information Models | by Stefan Profanter | Medium.” Accessed: Dec. 04, 2023. [Online]. Available: <https://profanter.medium.com/how-to-create-custom-opc-ua-information-models-1e9a461f5b58>
- [115] “UaModeler ‘Turns Design into Code’ - Unified Automation.” Accessed: Dec. 04, 2023. [Online]. Available: <https://www.unified-automation.com/products/development-tools/uamodeler.html>
- [116] “UA Part 4: Services - 4 Overview.” Accessed: Dec. 10, 2023. [Online]. Available: <https://reference.opcfoundation.org/Core/Part4/v104/docs/4>
- [117] “Gateway - Ignition User Manual 8.1 - Ignition Documentation.” Accessed: Jan. 15, 2024. [Online]. Available: <https://docs.inductiveautomation.com/display/DOC81/Gateway>

- [118] “Gateway Network - Ignition User Manual 8.1 - Ignition Documentation.” Accessed: Jan. 15, 2024. [Online]. Available: <https://docs.inductiveautomation.com/display/DOC81/Gateway+Network>
- [119] “Tag Providers - Ignition User Manual 8.1 - Ignition Documentation.” Accessed: Jan. 15, 2024. [Online]. Available: <https://docs.inductiveautomation.com/display/DOC81/Tag+Providers>
- [120] “Siemens - Ignition User Manual 8.1 - Ignition Documentation.” Accessed: Jan. 15, 2024. [Online]. Available: <https://docs.inductiveautomation.com/display/DOC81/Siemens>
- [121] “MQTT Modules - MQTT Modules for Ignition 8.x - Confluence.” Accessed: Jan. 16, 2024. [Online]. Available: <https://docs.chariot.io/display/CLD80/MQTT+Modules>
- [122] “MQTT Transmission - MQTT Modules for Ignition 8.x - Confluence.” Accessed: Jan. 16, 2024. [Online]. Available: <https://docs.chariot.io/display/CLD80/MQTT+Transmission>
- [123] “MQTT Engine - MQTT Modules for Ignition 8.x - Confluence.” Accessed: Jan. 16, 2024. [Online]. Available: <https://docs.chariot.io/display/CLD80/MQTT+Engine>
- [124] “Greenfield VS. Brownfield Smart Factory - Results Engineering.” Accessed: Feb. 27, 2024. [Online]. Available: <https://resultseng.com/greenfield-vs-brownfield-smart-factory>
- [125] “VDMA - Robotics - OPC Foundation.” Accessed: Jan. 24, 2024. [Online]. Available: <https://opcfoundation.org/markets-collaboration/robotics/>
- [126] N. Anumbe, C. Saidy, and R. Harik, “A Primer on the Factories of the Future,” *Sensors* 2022, Vol. 22, Page 5834, vol. 22, no. 15, p. 5834, Aug. 2022, doi: 10.3390/S22155834.
- [127] J. Lenz, E. MacDonald, R. Harik, and T. Wuest, “Optimizing smart manufacturing systems by extending the smart products paradigm to the beginning of life,” *J Manuf Syst*, vol. 57, pp. 274–286, Oct. 2020, doi: 10.1016/J.JMSY.2020.10.001.
- [128] R. Harik, Y. Shi, and S. Baek, “Shape Terra: mechanical feature recognition based on a persistent heat signature,” *Comput Aided Des Appl*, vol. 14, no. 2, pp. 206–218, Feb. 2017, doi: 10.1080/16864360.2016.1223433.

- [129] K. Xia, C. Saidy, M. Kirkpatrick, N. Anumbe, A. Sheth, and R. Harik, "Towards Semantic Integration of Machine Vision Systems to Aid Manufacturing Event Understanding," *Sensors 2021, Vol. 21, Page 4276*, vol. 21, no. 13, p. 4276, Jun. 2021, doi: 10.3390/S21134276.
- [130] M. Intizar Ali, P. Patel, J. G. Breslin, R. Harik, and A. Sheth, "Cognitive Digital Twins for Smart Manufacturing," *IEEE Intell Syst*, vol. 36, no. 2, pp. 96–100, Mar. 2021, doi: 10.1109/MIS.2021.3062437.
- [131] K. Xia *et al.*, "A digital twin to train deep reinforcement learning agent for smart manufacturing plants: Environment, interfaces and intelligence," *J Manuf Syst*, vol. 58, pp. 210–230, 2021, doi: <https://doi.org/10.1016/j.jmsy.2020.06.012>.
- [132] C. Saidy, "MULTIMODAL ROBOTIC HEALTH IN FUTURE FACTORIES THROUGH IIOT, DATA ANALYTICS, AND VIRTUAL COMMISSIONING," Doctoral Dissertation, University of South Carolina, Columbia, 2021. Accessed: Feb. 07, 2024. [Online]. Available: <https://www.proquest.com/docview/2572552671?pq-origsite=gscholar&fromopenview=true&sourcetype=Dissertations%20&%20Theses>
- [133] K. Xia, T. Wuest, and R. Harik, "Automated manufacturability analysis in smart manufacturing systems: a signature mapping method for product-centered digital twins," *J Intell Manuf*, vol. 34, no. 7, pp. 3069–3090, Oct. 2023, doi: 10.1007/S10845-022-01991-4/FIGURES/16.
- [134] M. Kirkpatrick, A. Brasington, A. D. Anderson, and R. Harik, "CREATION OF A DIGITAL TWIN FOR AUTOMATED FIBER PLACEMENT," in *CAMX 2020 Conference & Exhibition*, 2020, pp. 21–24. Accessed: Feb. 26, 2024. [Online]. Available: <https://nextusc.com/wp-content/uploads/2020/11/2020Kirkpatrick.pdf>
- [135] K. Xia, C. Sacco, M. Kirkpatrick, R. Harik, and A. M. Bayoumi, "Virtual comissioning of manufacturing system intelligent control," *International SAMPE Technical Conference*, vol. 2019-May, 2019, doi: 10.33599/NASAMPE/S.19.1403.
- [136] M. Kirkpatrick, D. Sander, F. El Kalach, and R. Harik, "Motion capture based calibration for industrial robots," *Manuf Lett*, vol. 35, pp. 926–932, Aug. 2023, doi: 10.1016/J.MFGLET.2023.08.012.

- [137] F. El Kalach, R. Wickramarachchi, R. Harik, A. Sheth, and A. Sheth, “A Semantic Web Approach to Fault Tolerant Autonomous Manufacturing,” *IEEE Intell Syst*, vol. 38, no. 1, pp. 69–75, Jan. 2023, doi: 10.1109/MIS.2023.3235677.
- [138] R. Harik *et al.*, “Analog and Multi-modal Manufacturing Datasets Acquired on the Future Factories Platform,” Jan. 2024, Accessed: Feb. 26, 2024. [Online]. Available: <https://arxiv.org/abs/2401.15544v1>
- [139] C. Saidy *et al.*, “Building Future Factories: A Smart Robotic Assembly Platform Using Virtual Commissioning, Data Analytics, and Accelerated Computing,” Jun. 2020, doi: 10.33599/NASAMPE/S.20.0051.