UNIVERSITÀ DI TRENTO
Dipartimento di Ingegneria Industriale

eit Digital MASTER SCHOOL

UNIVERSITÉ CÔTE D'AZUR

Master's Degree in

Mechatronics Engineering

FINAL DISSERTATION

CLOUD WORKFLOW CONTROLLER FOR IIOT SOLUTIONS IN ADDITIVE MANUFACTURING

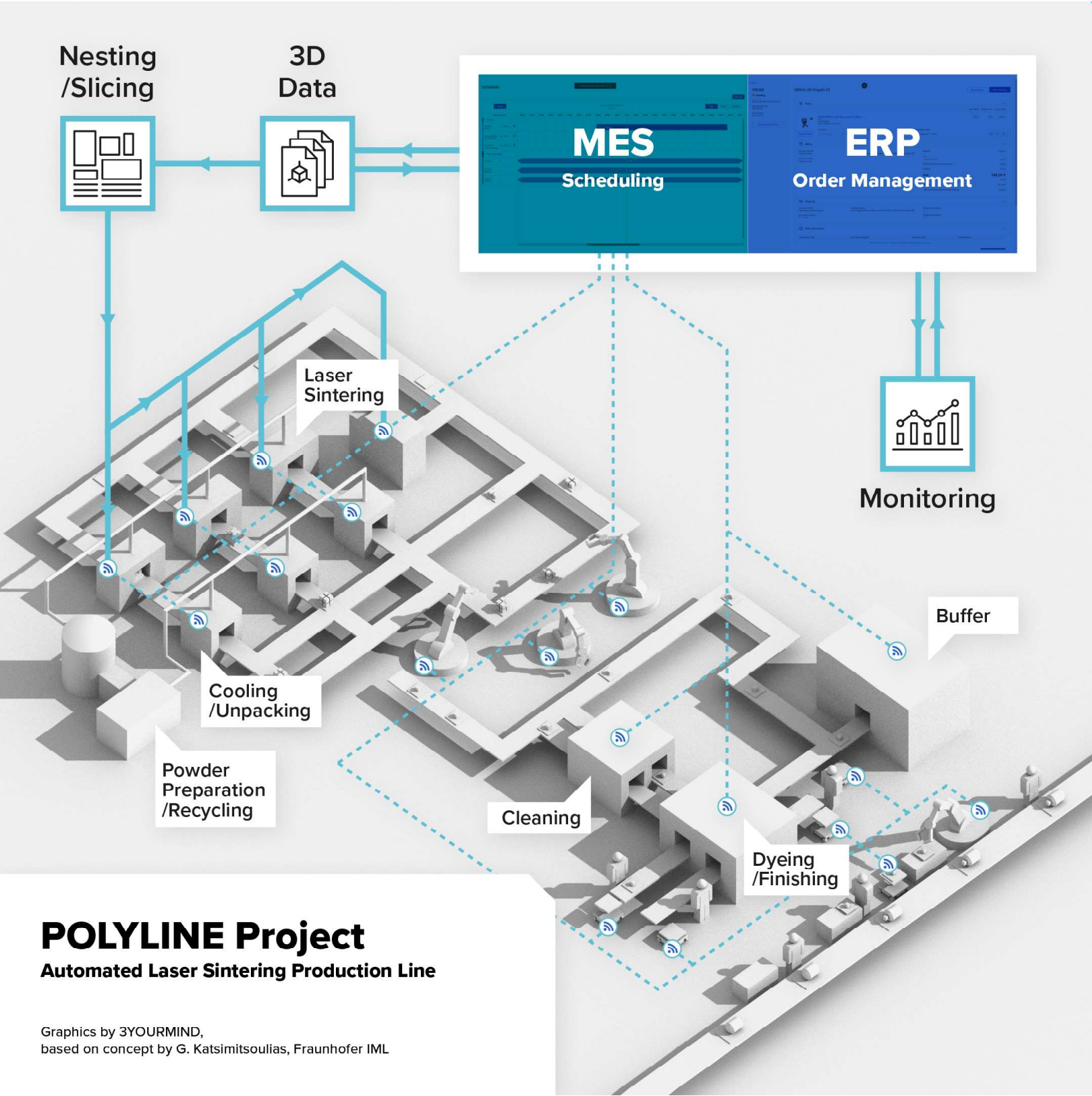
In the context of the POLYLINE project

Supervisor UniTrento
Daniele Fontanelli

Supervisor Company
Simone Dal Poz

Student
Zaida Brito Triana, [MAT. 230400]

ACADEMIC YEAR 2021/2022



POLYLINE Project

Automated Laser Sintering Production Line

Graphics by 3YOURMIND,
based on concept by G. Katsimitsoulias, Fraunhofer IML

Abstract

This master thesis presents the research and implementation of a cloud workflow controller for Industrial Internet of Things (IIoT) solutions in Additive Manufacturing (AM). This thesis has been carried out in the context of the POLYLINE project, which aims to automate the entire manufacturing process in the production of polymeric components within the additive manufacturing plant of BMW in Munich.

This research aims to explore suitable computational frameworks for creating a logic-based model for the project's production line. It analyzes common models like Finite State Machines(FSM) and Behavior Trees(BTs), with a focus on the latter's potential for organizing and managing complex processes and decision-making situations in manufacturing. By comprehensively examining BTs and other relevant models, this thesis seeks to identify the best approach for constructing logic-based models for production lines in Industry 4.0 in general.

Furthermore, this research underwent a comprehensive study of the principal communication protocols used in the Industrial Internet of Things. This evaluation was performed in order to provide a more comprehensive assessment of the proposed connections for the successful implementation of the workflow and give insights into the underlying automation functionality intrinsic to the project. While the primary means of machine communication employed in the project was based on the OPC UA protocol, a comparison with all the current state-of-the-art developments in communication protocols in this field was also undertaken. The results of this extensive analysis provided valuable insight into the most suitable communication protocols for future implementations.

Additionally, a detailed analysis of the initial data gathered was conducted to give insights into the initial outcomes of the implementation on-site. This analysis aimed to establish an initial baseline value for the Key Performance Indicators (KPIs) determined, with the goal of facilitating decision-making regarding future development.

The workflow controller has been tested in a mock environment prior to its implementation in the BMW plant. The tests have been focused on the ability of the system to support the workload generated by the production processes and deliver the intended result. The results indicated that the controller is capable of supporting the production workload.

This Master's thesis represents a contribution to the field of Industry 4.0, providing valuable insights and advancements in the application of automation technologies in Additive Manufacturing.

Acknowledgments

I would like to thank my mentor, Daniele Fontanelli from the University of Trento and my company supervisor Simone Dal Poz, as well as EIT Digital and my colleagues from 3YOURMIND for making this work possible. I would also like to thank my team colleague Denis Hezel for all his guidance.

I would like to thank my mother for all these years of immense support, for the breakfasts, the phone calls and the cafesitos. I would like to thank my father, despite his loss, his teachings always inspire me to pursue my dreams.

I would like to thank my friends from La Palma for their support and friendship in this complex volcanic year. I would also like to thank my friends from Valencia and Nice for creating a second home at university. And lastly, thanks to Francesco, for all the emotional support.

List of Abbreviations

Additive Manufacturing (AM)
Automated Guided Vehicles (AGVs)
Behavior Tree (BT)
Cloud Manufacturing (CM)
Enterprise Resource Planning (ERP)
Finite State Machine (FSM)
Flexible manufacturing systems (FSM)
Genetic Programming (GP)
Graphical User Interface (GUI)
Hierarchical Finite State Machine (HFSM)
Industrial Internet of Things (IIOT)
Key Performance Indicators (KPIs)
Industrial Message Queuing Telemetry Transport (MQTT)
Machine-to-machine (M2M)
Manufacturing Execution System (MES)
Non-player Character (NPC)
OpenAPI Specification (OAS)
Open Platform Communications United Architecture (OPC UA)
Probabilistic Finite State Machine (PFSM)
Programmable Logic Controllers (PLC)
Google Protocol Buffers (Protobufs)
Representational State Transfer (REST)
Supervisory Control and Data Acquisition (SCADA)
Universal Machine Type Interface (UMATI)

Contents

List of Figures

Chapter 1

Introduction

In the years to come, the number of additively manufactured products and parts will continue to grow, AM technologies are becoming increasingly important and have the potential to shape the world of production in the future [1]. Research into the optimal implementation of additive manufacturing technology has the ability to improve the manufacturing processes that run the general production globally, enabling more local production and reducing transport costs [2], as well as allowing for greater adaptability for OEMs and manufacturers [3].

AM integration into conventional lines of production in manufacturing can only be achieved to a limited extent at present, both vertically and horizontally [4] [5]. In this case, Horizontal integration could mean using 3D printing across different product lines or manufacturing processes to produce parts for various products within the same factory. Vertical integration on the other hand could mean integrating 3D printing at different stages of the manufacturing process, for example, using AM to create a prototype [6], then using traditional manufacturing methods to mass-produce the final product.

Initially, this limitation is thought to be a constraint due to the specific AM production steps (e.g., curing, post-processing, slicing, batch production time), but a closer examination reveals it's also related to the overall low-level implementation of automation techniques in additive manufacturing and post-production processes [5]. Due to this, longer production intervals are needed, and there is a great need for manual work involved not only to mass-produce but to prototype as well. Due to the gaps in the digital data chain across the horizontal process chain, there is insufficient monitoring of the processes, making it difficult to integrate them into relevant production controls. This impediment significantly reduces the full potential of additive manufacturing processes in the integration into existing series production and assembly lines. The state of the art in the industry shows that there is not enough automation in the additive process chain to fully exploit its potential,

as mentioned in [1]. The same remote technologies to enable automatization are still yet to be implemented in many industries, as evidenced by the findings of the study conducted by Seiffert et al. (2022) [7].

The current landscape presents numerous opportunities for Additive Manufacturing, as studied in [8] but until now, the research and implementation of digital solutions for managing production lines in Additive Manufacturing have been quite limited [9]. The integration and collaboration with other technologies, such as autonomous robots, cloud control, or IIOT, are shown to be key for the optimal implementation of Industry 4.0; the path to smart and flexible AM manufacturing goes through the development and integration with automation [5].

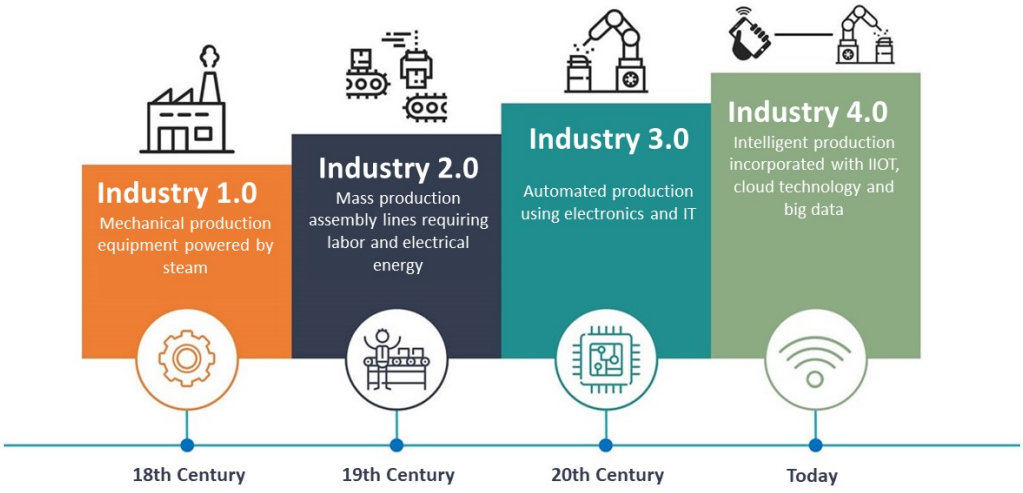


Figure 1.1: Industrial revolutions timeline

The rise of Industry 4.0, shown in Figure 1.1, and the increasing popularity of emerging technologies has created a unique space for the transformation of traditional manufacturing methods [10]. The current industrial value creation in early industrialized countries is influenced by the development towards Industry 4.0, the fourth stage of industrialization. This development presents substantial opportunities for achieving sustainable manufacturing through the use of AM technologies. The traditional centralized and monolithic production monitoring and control applications will be replaced by solutions that support the decentralized and connected vision of production and supply chain processes [11].

This research will conduct a state-of-the-art review of Industry 4.0, based on

recent advancements in research and practice, and provide an overview of the various opportunities for additive manufacturing in Industry 4.0.

Finally, it is worth highlighting that the COVID-19 pandemic has emphasized the need for rapidly and efficiently responding to real-time changes in manufacturing operations[12] [13], as the traditional manufacturing methods might be getting behind in their adaptability horizons [14]. Overall, the trend is clear: it is very important to adopt efficient, flexible, and agile manufacturing systems to overcome the challenges of real-time changes and volatility in demand. The adoption of advanced technologies foreseen in Industry 4.0, such as automation, AM, data management, and the Internet of Things, will help to improve manufacturing efficiency and reduce lead time in general.

1.1 Introduction to the Polyline Project

The POLYLINE project is a collaboration of German industry and research partners working to establish an automated factory for additive manufacturing of automotive parts. The objective is to fully automate the BMW AM production line, encompassing every aspect from CAD model to 3D printing and quality assurance. 3YOURMIND's Agile Manufacturing Suite will manage the entire AM workflow, and the resulting factory will serve as an example of Industry 4.0 for other future implementations. The goal of the project is to create an efficient and streamlined production line for end-use parts. Each partner will handle a specific aspect of the manufacturing process, from 3D printing to post-production and quality control. The project also aims to address issues such as long processing times, low degrees of automation for physical handling and transport processes and efficient integration of the entire additive production process into existing mass production.

The partners involved in this project include BMW, which owns the shop-flow where the machines are located; EOS, Grenzebach, Krumm-Tec, DyeMansion, and Nabetherm, which are the machine manufacturers; Additive Marking, which provides a software solution for applying serialization to the 3D geometry of the parts and 3YOURMIND that, based on partner documentation, is in charge to execute and implement the comprehensive workflow.

1.2 Objectives of the Thesis

One of the goals of this thesis is to give an overview of the state of the art in Industry 4.0, with a special focus on the domain of industrial automation applied to additive manufacturing. The objective is to conduct a comprehensive analysis of the current advancements and developments in this field. The research wants to provide insights that can be used to guide future development in the field of industrial automation in general, but particularly in additive manufacturing.

Furthermore, one of the objectives of this research work is to collaborate in the creation of a cloud workflow controller within the POLYLINE project, shown in Figure 1.2, which aims at the development of a fully digitalized additive manufacturing production line at BMW Germany. 3YOURMIND, one of the companies involved in this research project, is in charge of implementing and managing the entire AM workflow and its effective communication with the MES(Manufacturing Execution System) and the ERP(Enterprise Resource Planning), highlighted in Figure 1.2. Prior to this project, 3YOURMIND developed and commercialized an IIOT solution [15] for monitoring additive manufacturing production through IIOT. The general objective is to expand this solution's capacity to not only monitor the machines but also to remotely control and respond to multiple machines, potentially in different sites.

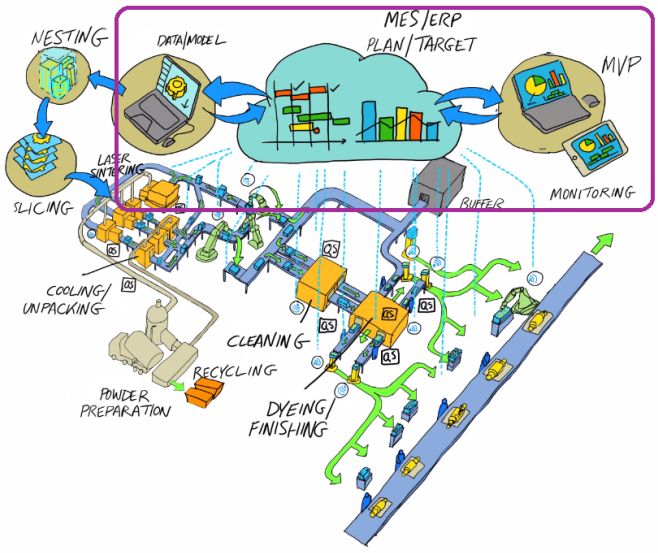


Figure 1.2: Schematic representation of the production line, highlighting the main focuses of the thesis.
©G. Katsimitsoulias, Fraunhofer IML (edited)

In order to achieve this goal, it is necessary to conduct research on the possible technologies that could be used to develop an automated production line. This research primarily concentrates on the workflow controller development and on exploring the implementation of a workflow trigger utility. The investigation specifically focuses on identifying the required models of computation that can be used to construct a logic-based model of the production line designed in the project, specifically researching on the advantages that Behavior Trees have offered for projects similar to this one in the fields of robotics, manufacturing, and gaming. In summary, the objective of this part of the research is to evaluate the strengths and weaknesses of various models, including BTs, to determine the most appropriate method for creating a logic-based model of the production line under consideration.

In the second segment of this dissertation, an in-depth examination will be carried out to analyze the communication protocols utilized in Industrial Internet of Things solutions in order to scrutinize the proposed connections for completing the workflow. The principal protocol implemented in the project is based on the OPC UA protocol; however, a comparative analysis with the current state-of-the-art advancements in this domain will also be executed. The outcomes of this comprehensive analysis will give insight into the most appropriate communication protocols for future research.

Since testing at BMW's facilities involves the collaboration of numerous companies, it was forecasted that testing on-site might not be carried out prior to the submission of this research, which is the reason why simulation has been considered the main testing tool of this research. The primary testing is implemented through a mock environment that will indicate if the controller is capable of supporting the production workload.

Lastly, an analysis of the initial data gathered will be carried out to gain a better understanding of the initial performance of the workflow implementation on-site. The objective of this analysis will be to identify the key performance indicators, such as the average duration of a manufacturing cycle and the average interval between error outputs, in the same way, the analysis will aim to locate the source of errors during cycles, all of this information could be used to reveal potential areas for improvement.

The insights collected from this analysis will serve as a resource for determining the overall system performance and will also allow for more informed decision-making regarding the optimal strategy for future development.

Chapter 2

Review of the Literature

The aim of this section is to establish the theoretical groundwork on which this research will be based. This section starts with a review of the current state-of-the-art developments in the field of automation in additive manufacturing, highlighting the current challenges and opportunities in the field in comparable applications. Following that, an analysis of the theoretical principles and concepts underlying Behavior Trees will be conducted, as well as a comparison with other similar models such as Finite State Machines and Hierarchical State Machines. This section will conclude with a discussion on the most common communication protocols currently utilized in Industrial Internet of Things solutions and their latest developments.

2.1 Automation in Additive Manufacturing

Additive manufacturing processes refer to techniques that build 3D objects by adding layer-upon-layer of material until the desired shape is achieved. The material can be plastic, metal, or other composites, and the process is controlled by computer software [16], [17]. AM includes various technologies such as 3D printing, laser sintering, and fused deposition modeling. It differs from traditional subtractive manufacturing methods, which involve cutting and shaping material from a larger block, in that technologies like CNC there are some advancements in automation mainly under the umbrella of Flexible manufacturing systems (FSM)[18].

It is crucial to analyze the advancements in automation within additive manufacturing [19] in relation to the broader trend of Industry 4.0, as the two are closely interrelated. The evolution of industrialization has established the foundation for the emergence of the fourth industrial revolution, commonly referred to as Industry 4.0, whose pillars are shown in 2.1. First introduced in 2011 with the goal of

promoting economic growth in Germany [20], this industry is characterized by the integration of cyber-physical systems (CPS). These systems, which constitute the third generation of control systems, consist of embedded computers and complex software applications that are interconnected through wired and wireless connections. These CPS possess the capability for autonomous decision-making and communication, with the purpose of improving industrial efficiency, productivity, safety, and transparency.

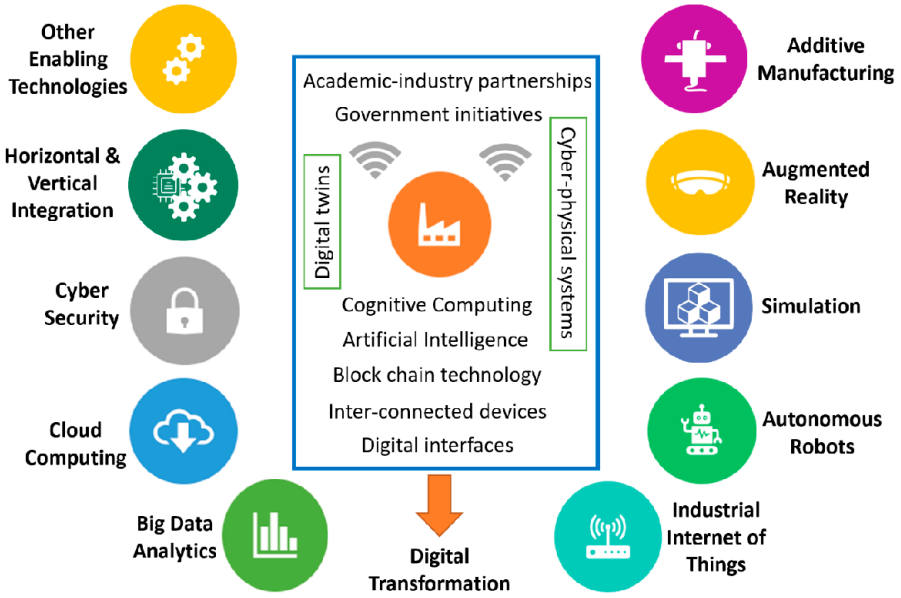


Figure 2.1: Technology enables of Industry 4.0 [5]

There is a significant correlation between the concept of Industry 4.0 developed in Germany and the Industrial Internet of Things concept. The term "Internet of Things" was first introduced in 1999 and encompasses interconnected devices present in various settings such as homes, businesses, and industries. Empirical studies have demonstrated, such as [21] that the implementation of wireless communication network intelligent automatic 3D printing machinery increases printing efficiency by 15% and saves economic cost by 20% . Additionally, there have been noteworthy advancements in the field of additive manufacturing, such as the incorporation of machine learning, as demonstrated in [22].

As seen in this article [23] advancements in automation technologies such as motion control components, robotic arms, and gantry robots will enable 3D printers

to overcome size constraints and increase the types of objects that can be "printed", creating new opportunities for manufacturers of these technologies. Studies have been conducted exploring the acceleration of production processes, with a particular emphasis on cloud-based additive manufacturing as a means of facilitating rapid development [24]. Recently, there has been an increase in collaboration between robotics and additive manufacturing for the purpose of enhancing production [25] [3]. Additionally, additive manufacturing has opened up promising sustainable prospects, as demonstrated by [8].

There is limited research available on the automation of additive manufacturing facilities, however, this study from 2015 [26] is relevant to the topic of the present investigation as a step into automation in AM. This study aimed to demonstrate the feasibility of automating the production planning of a 3D printing factory by modeling it as an optimization problem and testing it in a Shapeways factory, using EOS printers, as the ones from this research. The study seeked to create an automated process that could create batches, with the goal to reduce the amount of unused space in 3D printers, improving the 3D bin packing problem [27]. The result of this experiment showed a 10% increase in printer throughput compared to manually created batches.

2.2 Behavior Trees

Behavior Trees were developed with the intention of providing a method for developing the behavior of non-player characters (NPCs) in computer games because they usually need to respond to complex and uncertain situations in real-time, making decisions based on a set of rules and conditions. The concept of BTs was first introduced by R. G. Dromey in 2001 [28], with the publication of a paper defining the key concepts and applications of the model. However, one of the first significant implementations of BTs was in the development of the Halo 2 and Halo 3 video games [29], [30]. This demonstrated the potential of BTs in creating intelligent, autonomous systems that can react to changing environments.

As will be discussed in the following sections, in recent years Behavior Trees have been gaining attention for designing and implementing autonomous systems in computer programming and robotics [31].

Two Ph.D. theses, "Onboard Evolution of Human-Understandable Behaviour Trees for Robot Swarms" by Jones.SW [32] and "Behavior Trees in Robotics" by Colledanchise. M [33], will be used as the basis for discussion and analysis in this research. The latter thesis served as a precursor to the highly regarded book "Behavior Trees in Robotics and AI", [34] that will also be referenced in this study.

The example presented in this research paper was implemented using the "BehaviorTree. CPP" framework [35], written in C++ 17, and the "Groot" editor [36]. Both of these programs are maintained and regularly updated by Faconti.D and have been widely used in the implementation of BTs in autonomous systems. More insights into the advantages that these libraries offer will be discussed in Section 3.1.3.

2.2.1 Behavior Tree theory

Behavior trees are hierarchical structures used in artificial intelligence and robotics to determine system behavior. Nodes in a Behavior Tree represent specific actions or behaviors, and they are executed based on previous results and specified conditions. Behavior trees are ideal for specifying complex behaviors that involve decision-making and, furthermore, they allow for easy modification of the process through altering the tree structure. They are also simpler to understand and debug than other similar models.

Behavior trees are often used in robotics because they allow the robot to make decisions based on its current situation and the available information. For example, like in the POLYLINE project, a behavior tree node might specify that an AGV should pick the printed component if there is one available at the Transfer Station InOutFeed, better explained at 3.1.3.

Corresponding to the matter in this research, it is possible to define that a Behavior Tree is a hierarchical tree of nodes, where the leaf nodes correspond to executable programs that correspond to robot actions, such as picking up an part, stop a process, or perform a program. A behavior tree begins by running its root node, which sends out signals at a specific frequency called **ticks**. These ticks are then passed on to the children from the root node. A node can only be executed if it receives these ticks. When the root of a tree sends a tick signal, this signal it travels towards the leaves of the tree. Each tree node that receives the signal will then execute a callback function. This callback can return one of three possible results: success, failure, or running. If the running result is returned, it indicates that the action is still in progress and needs more time to be completed.

Tree nodes with children are responsible for passing on the tick signal to them. The rules for when and how many times the tick is passed down to the children may vary between tree nodes. The actual commands of the behavior tree are found at the leaf nodes, and it is there that the tree interacts with the rest of the system. Commonly, these leaf nodes are Actions nodes.

2.2.1.1 Common node types

A review of the most commonly used node types in industry projects and research is presented below.

Outline of the most common control flow nodes:

Sequence nodes: A sequence node is the most common type of node in a behavior tree. It is used to execute a series of actions consecutively. When a sequence node with more than one child receives a tick, it will first tick its first child; if that child returns a result of success, the sequence node will then tick its next child, continuing this process until all of its children have been ticked.

The sequence node will return a result of success if all of its children return a result of success. If any of the children return a result of failure, the sequence node will immediately return a result of failure and stop executing the behavior tree. Above, the pseudo code representation that elucidates the underlying logic of the node. This pseudo code serves as a guide for understanding the functionalities and decisions made by the node in executing its tasks.

Algorithm 1 Sequence node

```
1: for i ← 1 to N_child do
2:     childReturn ← Tick(C_i)
3:     if childReturn = RUNNING then
4:         return RUNNING
5:         if childReturn = FAILURE then
6:             return FAILURE
7:         end if
8:     end if
9: end for
10: return SUCCESS
```

Fallback or Selector nodes: The series of actions specified by these nodes is ticked in order. If a child returns failure, it ticks the next one. If success or running is returned by one child, the selector does not tick any more of its children, and the same result is returned. In the event that all of the actions fail, a failure result is returned by the selector node. Above, the pseudo code representation that explains the underlying logic of the node.

Algorithm 2 Selector or Fallback node

```
for i ← 1 to N_child do
    childReturn ← Tick(C_i)
    if childReturn = RUNNING then
        return RUNNING
        if childReturn = SUCCESS then
            return SUCCESS
        end if
    end if
end for
return FAILURE
```

Parallel nodes: These nodes indicate that a series of actions should be carried out at the same time. If any of the child nodes are still in progress, the node will return a running status. If the number of child nodes that have successfully completed is above a certain threshold, which is set by the user, the node will return a success status. If it is below the threshold, it will return a failure status. Although they were not used in this research, parallel nodes are mentioned here for completeness.

Decorator nodes: These nodes have a single child node, and they modify the return status of the child node according to a rule specified by the user. They also selectively execute the child node based on a predetermined rule. In every case, if the child node returns a running status, the decorator node will also return running. Algorithm 3 presents the pseudocode of the commonly used decorator node *RetryUntilSuccesful*.

Algorithm 3 Decorator node : RetryUntilSuccessful with N attempts

```
1: for i ← 1 to N_child do
2:     for n ≤ N do
3:         if childReturn = FAILURE then
4:             n = n + 1
5:             if childReturn = SUCCESS then
6:                 n = N
7:                 return SUCCESS
8:             end if
9:         end if
10:    end for
11: end for
12: return FAILURE
```

Outline of the general characteristics of control execution nodes:

Action nodes: Action nodes are the most common type of leaf node in BTs, and they represent a specific action or behavior that should be performed. When it receives a tick, an action node will be executed and, if the action is completed successfully, it will return a success status. If the action fails, it will return a failure status. While the action is in progress, it will return a running status. An action node may read from the blackboard, better described in Section 2.2.1.2, or modify the values of certain variables by writing to a key in the blackboard.

Condition nodes: Control execution nodes determine the success or failure of a particular action based on the evaluation of a specific condition. They are frequently used with decorator nodes to specify the circumstances under which a specific behavior should be executed. It is important to note that a control execution node will never return a running status and also, they may read from the blackboard, but they cannot modify its contents.

The table above 2.1, inspired by [34], summarizes the characteristics of the most common node types

Node type	Symbol	RUNNING	SUCCESS	FAILURE
Sequence		If one child returns RUNNING	If all children return SUCCESS	If one child returns FAILURE
Fallback		If one child returns RUNNING	If one child return SUCCESS	If all child returns FAILURE
Parallel		Any other situation	If N children return SUCCESS	If > n - N return FAILURE
Decorator		Custom	Custom	Custom
Action		During completion	On conclusion	If unfeasible
Condition		Never	If true	If false

Table 2.1: Most frequently encountered node types in a Behavior Tree, outlining their main characteristics.

2.2.1.2 Blackboard

In a behavior tree, leaf nodes interact with the environment through the blackboard. A blackboard is a shared data structure that allows nodes within a Behavior Tree to communicate and exchange information with each other. It serves as a centralized storage location for data that can be accessed by the nodes. This allows for the Behavior Tree to make informed decisions based on the collective information available on the blackboard as can bee seen for example in studies like [37].

Blackboards are are a key component in behavior tree logic because they allow the nodes to share information and coordinate their actions without the need for direct communication. This is a crucial characteristic of BTs because it makes possible to get informed about changing circumstances or react to them by accessing the data stored in the blackboard, this mechanism will be better explained in later Sections like 3.1.3. Input ports in a node are capable of accessing information stored in entries in the Blackboard, whereas output ports can create new entries in

the Blackboard. As a result, output data from one node can be used as input for another node in the behavior tree, the value of a blackboard entry can be access through a key.

In the shapes project [38], it is possible can see a basic example of the use of blackboard when the robot outputs a string with the name of the identified person after recognizing it.

Blackboards are one of the key elements in the real implementation of behavior trees; it is possible to see it for example in [39] in the explanation of the controller of its system "A behavior tree consists of a tree of nodes and a blackboard of variables which comprise the interface between the controller and the robot. At every controller update cycle, the tree of each robot is evaluated, with sensory inputs resulting in actuation outputs. Evaluation consists of a depth-first traversal of the tree until certain conditions are met. Each agent has its own blackboard, state memory and tree."

2.2.1.3 Challenges with Behavior Trees

Some of the limitations or challenges of BTs include finding the right balance between complexity and simplicity since overly complex trees can be difficult to understand and modify, while overly simple trees may not capture all necessary behaviors. To determine the right level of granularity, or detail, for each node, as well as the appropriate conditions for branching between nodes can be particularly challenging in complex, dynamic environments where the conditions for behavior may change rapidly.

Additionally, between the challenges of BTs it is possible to highlight that they are less mature and the software and documentation that can be found for this purpose, while useful,is still behind other more mature models, like FSM.

2.2.2 Behavior Trees vs Finite State Machines

FSMs have been for long time the standard choice for constructing structures for task switching, in 2013, Klockner highlighted the current prevalent use of Finite State Machines at that time as a means of managing the diverse capabilities of an Unmanned Aerial Vehicle (UAV), quoting in his paper "The state-of-the-art approach for managing different capabilities of a UAV is to use Finite State Machines." [40]

In a comprehensive manner, FSMs are a computation model that can be in only one state at any one time. The state changes when specific conditions are met,

which are usually linked to sensor inputs. The outputs of actuators are usually dependent on the current state.

Despite the multitude of possibilities that finite state machines offer, it has been noted that they encounter difficulties in satisfying two essential characteristics that are often sought-after in autonomous agents: the ability to exhibit both **reactive** and modular capabilities.

The term reactive refers to the ability to rapidly and effectively respond to changing circumstances. This can manifest in various ways as for example when a new obstacle appears on the path of a robot and reroutes to avoid a collision. The **modularity** allows for a more hierarchical and adaptable approach in designing, which can lead to more flexibility and ease of modification of the agent's behavior.

As explained in [34], the problem lies in FSM's trade-off between reactivity and modularity. This compromise can be comprehended by referencing the traditional Goto statement, heavily criticized by Edsger Dijkstra [41], that was utilized in early programming languages. The Goto statement serves as an example of a one-way control transfer, in which the execution of a program diverts to another section of code and continues from that point. In contrast to one-way control transfers, contemporary programming languages tend to favor two-way control transfers, as exhibited in constructs such as function calls. In order for the system to be reactive, many transitions between components are necessary. However, a high frequency of transitions results in an abundance of one-way control transfers. For instance, if a component is removed, every transition to that component must be updated, affecting lastly the modularity. BTs, on the contrary, utilize two-way control transfers regulated by the internal nodes of the tree, which has been shown to mitigate this trade-off.

As the number of capabilities or inputs that a finite state machine is required to process it increases the complexity of the FSM, also increases in terms of the number of states and transitions. This results in a scalability issue, as the FSM becomes increasingly difficult to design, analyze, understand, maintain, and verify as the number of states and transitions increases, making it challenging to ensure that the FSM is working as intended.

It is also worth noting that behavior trees offer advantages over finite state machines in terms of reusability. The hierarchical structure of behavior trees enables the reuse of subtrees, allowing for more efficient creation of new behaviors or the extension of existing ones. This modular design allows for a more streamlined development process, as it enables developers to easily add or modify behaviors without having to make extensive changes to the overall structure of the system.

It can be highlighted that while behavior trees provide an advantage over finite state machines, it has been demonstrated through the research of Colledanchise [2017] and Jones [2020] that both BTs and FSMs are mathematically equivalent in

terms of their capabilities.

The validity of these advantages has been substantiated through various contemporary studies, including [38] [42], and more insights on the topic will be examined in greater detail within 3.1.3 Section.

2.2.2.1 Hierarchical Finite State Machines

Hierarchical Finite State Machines (HFSMs), were created to address some of the limitations of traditional FSMs. In an HFSM, a state can contain one or more sub-states, a state that contains two or more states is known as a super-state. In an HFSM, transitions between super-states are referred to as generalized transitions, the purpose of these transitions is to reduce the number of transitions needed by connecting two super-states instead of connecting a larger number of sub-states individually.

One disadvantage of using Hierarchical Finite State Machines over Behavior Trees is the increase complexity, since as the number of states and transitions in an HFSM grows, the connections in the system can become increasingly complex and difficult to understand. This can make it harder to maintain and update a manufacturing workflow, especially when new machines are added or when processes are removed.

Additionally, the tree structure of BTs also allows for a more modular approach the design of the system, making it easier to add, remove, or modify individual behaviors without affecting the rest of the system. Furthermore, BTs are well suited for parallel execution, where multiple tasks can be run at the same time, whereas HFSMs are not able to perform parallel execution.

It is worth noting that despite these potential disadvantages, HFSMs can still be a suitable choice for certain implementations depending on the requirements of the system, but for more complex or dynamic systems, like the one presented in this research, BTs are often considered for offering a more flexible and scalable option.

2.2.3 State of the art in Behavior Trees

Even though BTs have been around for more than a decade now, not too many years ago the complaint was that in the field of Behavior Trees "The available literature lacks the consistency and mathematical rigor required for robotic and control applications" as stated in [43]. Since then, attempts were made to standardize the basis of behavior trees, as seen in [44]. The purpose of this review is to provide a comprehensive overview of the current state of the art in the application of behavior trees in autonomous systems reviewing two recent studies. There is not much literature on similar applications to the one in this research that uses BTs, but these studies review the use of BTs for complex distributed systems, in settings that share characteristics with this project.

In this research conducted in 2021, Legarda.H [45] studied a methodology for the collective transport of heavy objects through the utilization of an industrial swarm in a simulated setting is presented. The method was developed and tested in a simulated environment, using a combination of freely available open-source libraries. The robots employed controllers that incorporate a behavior tree architecture, which were optimized using genetic programming (GP) techniques to generate actuator commands. Coordination was achieved through a decentralized negotiation strategy that employed direct communication. The findings indicated that the genetic programming algorithm is capable of generating controllers that are able to safely lift and convey multiple loads concurrently towards a designated area, referred in the project as nest region. However, further refinement would be needed to implement these advancements in a real-world environment in order to assess its sensitivity to the reality gap.

The second research [39], linked as well with another earlier publication [46], puts the focus on the issues of designing optimal and readable systems using BTs. In order to design swarm robot systems with desired collective behavior, the challenge lies in designing controllers for individual robots that will produce the desired communal behavior through interaction. The current solutions studied for the mentioned research often relied on offline automatic discovery using artificial evolution of robot controllers, which are then transferred to the swarm, but that approach had limitations such as the need for additional supporting infrastructure for evolution and communication and also the evolved controllers being opaque and difficult to understand which could compromise safety and explainability. This research addresses both these issues by using behavior trees. Behavior trees were used as the individual robot controller architecture with great results, focusing on them being modular, hierarchical, and human-readable. Automatic tools were developed to simplify large evolved trees, making it possible to understand, explain, and improve the evolved controllers.

The research highlights three main topics to research in the future in the field of BTs; firstly, adaptability represents a crucial aspect to be considered,meaning for the system described to posses the ability to better adapt its behavior in response to changes in the environment. Another important area of investigation is the reality gap and the impact of the chosen architecture. Additionally, expanding the system to operate in three-dimensional environments is also worth exploring, since the research was made in a two-dimensional setting. The videos of the implementation of this research can be consulted.

2.3 Communication Protocols in IIOT

The current manufacturing schema finds itself on the cusp of a new epoch. As the processing power of industrial embedded devices continues to increase, the creation of many innovative applications that have the potential to alter the way manufacturing and consumption is performed is on the horizon. These advancements, encapsulated inside the concept of Industry 4.0 depend largely on the effective communication between devices and are covered under the term Industrial Internet of Things. One of the obstacles to achieving effective communication is the existence of many diverse protocols developed in different domains, as stated in [47]. This section will focus on the protocols that are most commonly used in industrial environments, particularly in the area of additive manufacturing, such as OPC-UA (in conjunction with Umati), MQTT, and HTTPS (along with REST API).

The objective of this section is to provide a comprehensive examination of the underlying principles behind the mentioned communication protocols, which will be then subjected to comparative analysis in the experimental section of this report. All of this is aimed with the intention to provide the theoretical foundation necessary to comprehend the conceptual framework within which this study is developed.

2.3.1 OPC-UA

Given that OPC UA or Open Platform Communications Unified Architecture [48] is the primary communication protocol employed in this project and the only one that has been evaluated on-site, it is coherent to initiate the communication protocols analysis by focusing on it. By establishing the fundamental theoretical concepts of OPC UA, it will be possible to examine in the following sections the advantages, challenges, and future prospects of this protocol in the field Industrial Internet of Things.

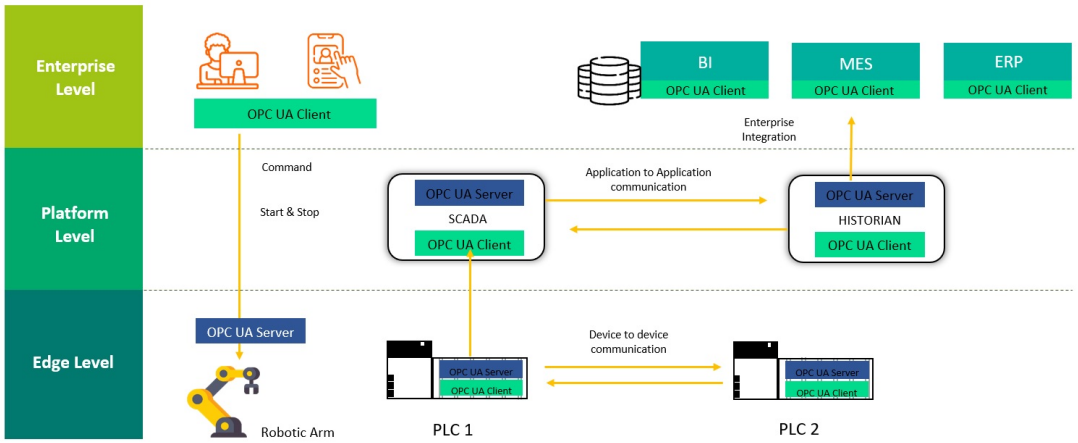


Figure 2.2: Schema of an OPC UA implementation in industry [49]

OPC UA is considered as a key enabler of Industry 4.0, as stated in Bauer et al. (2021) [50] because it makes possible to exchange data and information in real-time between different Industrial Ethernet protocols, allowing different devices and systems to communicate with each other even if they use different communication protocols, in a vendor-neutral way. As illustrated in Figure 2.2 and also explained in this recent study [49] that analyses the possibilities of this communication protocol in Industry 4.0, OPC UA can be used at different levels of the three-tier architecture. In the three-tier architecture, the first tier is responsible for data acquisition, the second tier for data processing and storage, and the third tier for data presentation and visualization. This study also highlights that OPC UA is designed to provide efficient communication horizontally between different industrial devices and systems, including Programmable Logic Controllers (PLCs), Supervisory Control and Data Acquisition (SCADA) systems, and edge gateways, this interopteroperability is the enabler of the use of OPC UA in the called M2M Machine to machine (M2M) communication that is is becoming increasingly important in this new era of manufacturing[51] [52].

For a while now, OPC UA has been regarded as a crucial communication protocol in the industrial automation field [53]. However, there are concerns regarding the limited adoption of OPC UA in embedded field devices, such as sensors and actuators, due to the restrictive memory and power requirements of software implementations.

The software framework developed by 3YOURMIND and utilized in the present study is based on open62541 [1], which is a dedicated open-source software repository

[1] https://www.open62541.org/

for OPC UA implementations in general. This library allows for reading the current value of a node and monitoring its values. The original library was implemented in C but in the 3YOURMIND implementation, it has been encapsulated within C++ code to enhance its usability. As an example, in the 3YOURMIND Machine Connectivity implementation the values from the machines from the workflow can monitored through a callback function referred to as `"LoggingNodeValueAssigner"`, the value of the node is subsequently stored in an appropriate variable

`Polyline::CurrentNodeValue robot_cell_handling_robot_ready`,

that then can be monitored by the workflow controller . In the Polyline project in a previous step in this research, the necessary functions were developed to establish the monitoring of nodes in the OPC UA server, and their corresponding callbacks were linked to the relevant assigner. This step is key in enabling the value stored in the designated assigner to be retrieved and subsequently saved in the blackboard entry in the Behavior Tree.

As a hint into the real application of OPC UA communication it is important to highlight that by convention, the default port assigned to OPC UA is 4840. This port is used for transmitting and receiving data in OPC UA communications. It is worth noting that although the default port is 4840, it is possible to configure OPC UA to utilize a different port if necessary. However, if a non-standard port is used, it may require additional configuration on the network infrastructure.

Within the framework of this research, it is worth dicussing some of the OPC UA implementations developed by the involved manudacturers. Several companies, including DyeMansion and Krumm-tec, have opted to implement an OPC UA server for the machines using Siemens hardware and software. Siemens has developed an extensive platform that offers native support for OPC UA-based applications, which includes a range of hardware products such as the SIMATIC Edge Device, communication modules, and software engineering tools like the TIA Portal [54], as well as OPC UA server and client libraries.

2.3.1.1 UMATI

UMATI (Universal Machine Type Interface)[55] is a standardized protocol for communication between industrial machines and controllers. It was developed as a collaboration between the OPC Foundation, the VDW, the German Machine Tool Builders‘Association and the VDMA, the German Engineering Federation, with the goal of creating a universal interface for machine communication that is open, standardized, and easy to use. The UMATI standard uses OPC UA, better explained in the section above, for the underlying delivery method, which is platform-independent, secure, and offers reliable communication between machines

and controllers in industrial environments [52] UMATI enables interoperability between different machines and controllers, regardless of the manufacturer or communication protocol they use. This makes it easier to integrate new machines into production processes and to connect different types of equipment together.

UMATI aims to establish a standardized set of meanings to be integrated into the existing OPC UA information model by designing a consistent data structure for both servers and clients that builds upon the pre-existing OPC UA format. This strategy reduces risk for vendors, since OPC UA has already undergone extensive testing and validation as a data standard, as it can be seen in studies like [56] and [51]. Once the UMATI standard is fully defined, it will simplify the process of connecting disparate machine types and applications and it will also streamline the connection of machines from multiple vendors, as well as enable easy integration with other OPC UA implementations, even if they have not yet implemented UMATI.

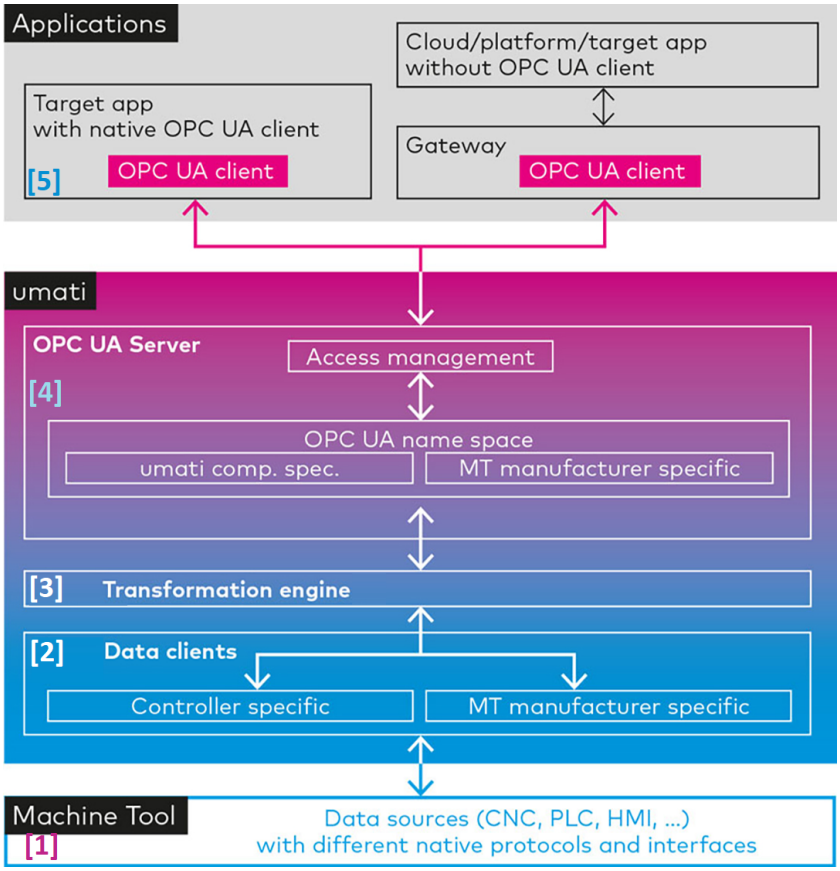


Figure 2.3: Configuration required to make a machine compatible with UMATI Edition based on the diagram specifications published in VDW

Figure 2.3 is a schema that explains the configuration required to make a machine compatible with UMATI. The structure is that the operating information is stored in the control device and software, this conforms the Data sources (1), where the information of the machines come from. The data clients (2) retrieve the operating information from the data sources, and then the transformation engine (3) converts the data to the UMATI data format. The data is then sent using the OPC-UA communication standard through the OPC UA server (4). The data clients, transformation engine, and OPC UA server are the parts that make up the UMATI gateway, which allows to connects machine applications to UMATI. The OPC UA client (5) connects systems and software applications to the UMATI gateway via a network. The system allows machine application to be accessed using the same UMATI data format, in a vendor independent way.

Considering the context of the project, it is worth noting that 3YOURMIND has played an active role in the UMATI working group since August 2019. In Machine Connectivity the implementation of the project adopts the UMATI schema for structuring data, the connection between the machines and the MES is established through OPC UA. The information exchanged is serialized under the file umati.proto, using the proto buf format, which is an encoding format developed by Google that will be better explained in Section 3.1.1

2.3.2 MQTT

MMQTT(Message Queuing Telemetry Transport) [57] is a lightweight message protocol that uses a subscription-publishing model, as shown in Figure 2.4 to allow publishers to send messages to a server that forwards them to subscribers, avoiding point-to-point connections between subscribers and publishers. A publisher is a device or application that sends messages to a broker, which acts as an intermediary between publishers and subscribers. A subscriber is a device or application that receives messages from the broker. They can also filter messages based on certain criteria to receive only relevant information.

The broker is the central hub of the MQTT system, responsible for receiving messages from publishers and forwarding them to subscribers. It is also responsible for enforcing security and access control policies, and for managing message queues when subscribers are not available.

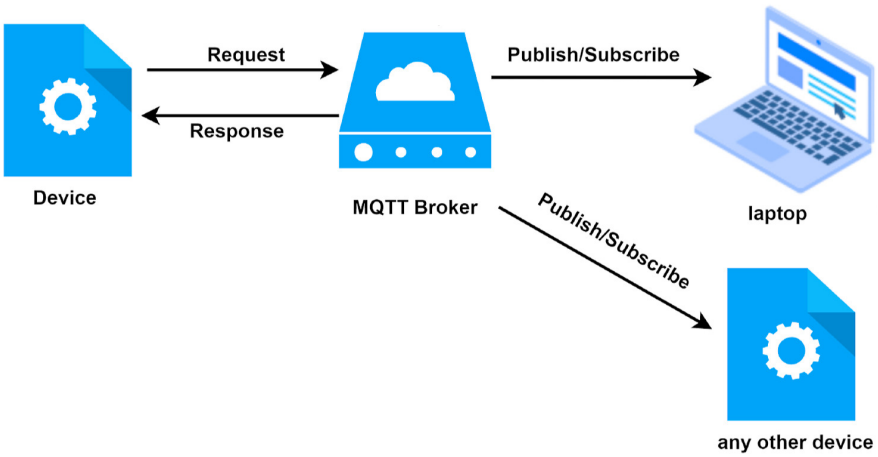


Figure 2.4: MQTT Publish / Subscribe Architecture
Edited from [58]

As explained in the theory foundation of [59] ,MQTT is a messaging protocol that offers flexibility and efficiency in exchanging data in various formats like JSON, XML. The protocol offers three different levels of Quality of Service (QoS) to ensure the message delivery.

MQTT is a communication protocol widely used for low-power devices that need to send information to a server[60]. It is popular for its low bandwidth consumption, high performance, and low latency, making it an ideal choice for IoT and M2M communication. MQTT has become one of the standards in this type of implementation, with many real-world applications utilizing it, such as those demonstrated in Fisher's work [61]. It is compatible with various devices, including Arduinos, Raspberry Pis, and commercial home automation solutions, which speaks to its flexibility, this characteristics has been also review in studies like [62] that tried provide a common schema of an MQTT implementation.

2.3.3 Http

This protocol is the fundamental client-server model protocol used for the Web [63], and the one most compatible with existing network infrastructure, used by the web developers on a daily basis. The Hypertext Transfer Protocol (HTTP) is a commonly employed communication protocol for the transfer of data across the Internet. Within the field of the Industrial Internet of Things, HTTP can be use to facilitate communication and data exchange among devices and systems within a network. One implementation of HTTP in IIoT applications is through the

utilization of Representational State Transfer Application Programming Interfaces (REST APIs). REST APIs are web services that conform to the REST architectural style and use HTTP to transmit and receive data.

The HTTP protocol serves as the underlying communication mechanism for REST APIs, with HTTP methods such as GET, POST, PUT, and DELETE utilized for sending requests and receiving responses between the client and server. REST APIs typically utilize HTTP status codes to indicate the success or failure of a request and HTTP headers to carry supplementary information, such as authentication tokens, content types, and other metadata.

In the following section, we will delve further into the topic of REST API, as it serves as the framework in which we have conducted our tests for the implementation of a potential triggering mechanism for the machines in the production line of the project.

2.3.3.1 REST API

As explained in the section above, it is imperative to discuss the underlying theory of the REST API, as the framework for the trigger mechanism that was researched using Markforged 3D printers. These printers are equipped with the Eiger API V3 [64], which operates on the REST API architecture. As it will be further elaborated later on Section 3.1.1, the outcome of this implementation only allowed the retrieval of reports from the Markforged printers. The Markforged company provides a REST API for their industrial 3D printing systems. This API allows developers to integrate the functionality of the 3D printing systems into other applications and automate various tasks such as directing printing and monitoring the print progress.

The REST API uses HTTP requests to send and receive data and is designed to be easy to use and flexible for integration with a wide range of systems. The REST architecture, was introduced by computer engineer Dr. Roy Fielding in his doctoral thesis in 2000, [65]. REST is a popular method for connecting components and applications within a microservices architecture due to its flexibility and autonomy for developers, as described by Schiekofer et al. (2018) [66]. It uses either JSON or XML to send and receive data and is based on the client-server model, where the client triggers actions on the server and the server reacts. There has been a significant amount of research and technological progress in the implementation of REST APIs in different contexts. One example is the use of REST APIs in Android mobile apps, as studied by Abdellatif et al. (2020) [67]. Another example is the use of REST APIs in cloud testing, as explored by Wolde Boltana (2021)[68]. In the field of 3D printing, REST APIs are utilized to transfer 3D models, automate printing processes, and allow for remote control of 3D printers.

Chapter 3

Methodology and results

The present study implementation was conducted within the framework of Machine Connectivity from 3YOURMIND [15]. This IIOT solution is based on the utilization of the OPC UA communication standard along with UMATI and will be further explained in later sections.

First, a research on the possible models of computation that will be used to model the logic of the production line has been conducted. Once the model is chosen, it will be necessary to extend the available debugging tools as well as implementing a trigger utility to start the workflow, this trigger utility will also be used for pulling production reports from a Markforged machine using REST API, a standard for internet data exchange, this data transfer with a real machine is a key component to deepen the understanding of IIOT solutions in this research as well as to simulate and test the project beforehand.

After developing the trigger utility, the next step will be to continue with the development of the workflow controller using Behavior Trees available libraries to implement the machines belonging to the production line of the project. To ensure the achievement of the determined objectives in this research simulation was chosen as the primary testing method. The software development approach chosen in this research is Test Driven Development through automated mock testing with gtest. This approach has proven successful over time and is now considered to be a standard [69].

In addition, the complete POLYLINE project will be integrated and tested in the BMW production structures by 3YOURMIND. Even though simulation was chosen as the primary testing method for this research, in Section 3.1.3.10 the industrial testing process will be discussed to some extent, highlighting some of the advantages that Behavior Trees have shown in the testing, the data collected from the on-site testings will be analyzed in order to derive Key Performance Indicators that provide a general understanding of the performance results of the project.

3.1 Project outline

Before embarking on a discussion of the different implementations and testings that have been conducted, it is imperative to first gain a deeper understanding of the fundamental design of the system. This is because the backbone of the system's design serves as the foundation for all subsequent developments and is critical to the system's overall effectiveness and efficiency. The schematic representation of the system's functionality, as depicted in Figure 3.1 , provides a comprehensive illustration of the operations that take place within the framework of the project. It is through this representation that the underlying design principles, communication channels, and component interactions can be understood.

The trigger utility, developed as part of this research, is situated within the Manufacturing Execution System framework. Its purpose is to transmit the required files and initiate the execution of the workflow. The trigger utility plays a pivotal role in ensuring that the workflow is executed in a timely and efficient manner, serving as the starting point for the various processes and operations that take place within the system, meaning it first send the files and starts the workflow. The input signals the aggregator receives are then passed for the activation of Behavior Trees.

Upon activation, the Behavior Trees then execute their sends commands to the machines (like Falcon, Grenzebach, EOS,. . .)based on the logic and decisions made by the tree's nodes, which are programmed to perform specific actions or make decisions based on inputs from machine servers, in this case OPC UA servers. After receipt of the commands, the machines execute the instructions and send the results back to the Behavior Tree, which reacts accordingly. The aggregator then forwards the results to the Manufacturing Execution System.

Upon observing Figure 3.1, it is pertinent to clarify the role and functionality of the *"drivers"*, despite they fall out from this thesis's scope, they are necessary to understand the complete logic of the project. Conventionally referred to as *drivers*, they are essentially a constituent element of the Machine Connectivity system developed by 3YOURMIND, and are integrated into the aggregator subsystem. These drivers are responsible for translating data received from the different communication protocols nodes, like for example OPC UA node into a schema that is shared among all drivers. They connect to the machine, gather relevant information, and translate it into the UMATI data structure. The ultimate objective is to transform the information from the printer into a schema that conforms to the UMATI standard. The resulting data, structured as per UMATI specifications, is transmitted to the Manufacturing Execution System using HTTP, with the data being encoded into a protobuf format.

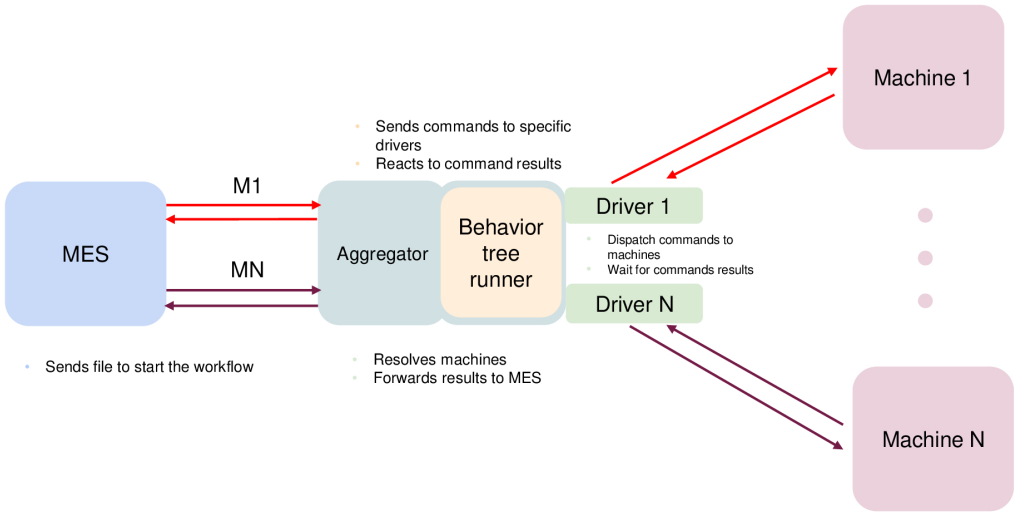


Figure 3.1: Schematic representation of the system functionality

3.1.1 Trigger utility - Markforged reports

At the outset of this research project, one of its objectives was to develop a trigger mechanism that would enable the activation of the machines involved in the workflow, such as starting job printers and postprocessing routines. The primary aim of this initial step was to gain a deeper understanding of the communication framework between the Manufacturing Execution System and the industrial machines and to identify areas for improvement. The trigger utility was envisioned to be responsible to start some of the jobs in the workflow, and it was tested using a Markforged printer. The initial plan for this implementation was to trigger a printer job, generate production reports, and save them using a Markforged machine. The purpose of this implementation was to establish a framework for the eventual on-site deployment of the Polyline project.

The trigger utility would act as the starting point of the production process, initiating the workflow and setting the wheels in motion for the rest of the tasks to be executed, based on the deployed framework in this section, the MES would send a switching on signal and then when the correct Behavior Tree node would have been triggered, it would have sent the signals to the machines included in the project. This particular deployment was also designed with the intention of

extracting production reports from the Markforged machine, which would provide a comprehensive overview of the production process. The schema of functionality is shown in Figure 3.2

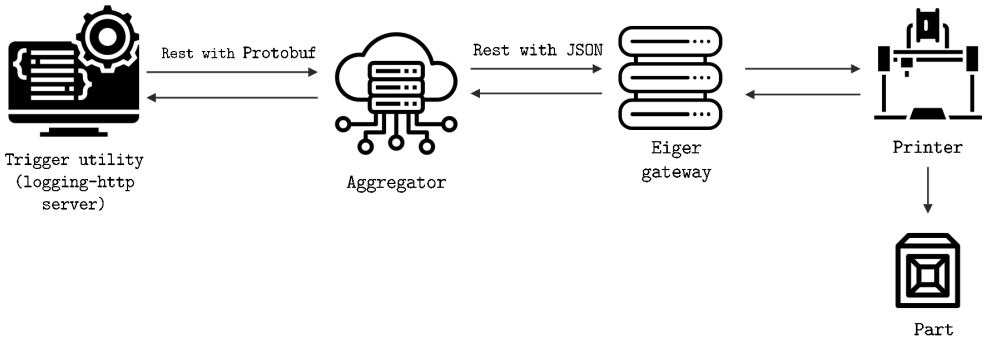


Figure 3.2: Communication architecture of the implementation with the Markforged printer

The Eiger API, developed by Markforged, is the software development tool that allows to integrate Markforged 3D printing solutions into custom applications, workflows, and systems, including the one employed in this study. The API is described by an OpenAPI Specification (OAS), a standard for describing and documenting RESTful APIs. Talking about the main characteristics of this REST API implementation, the Eiger API is structured around resources and collections of resources that are accessible through unique URIs, the interaction with these resources is performed using standard HTTP methods and primarily uses JSON to exchange data between the client and server. Access to the Eiger API is secured through authentication, which is used to identify the API client and verify their authorization to use the API. The authentication process is accomplished through HTTP Basic Authentication and the use of API keys, this keys consist of two components: an API Access Key and an API Secret Key.

Even though the Eiger API primarily uses JSON to exchange data between client and server it possible to use Protobuf (Google Protocol Buffers), a binary serialization format developed by Google, to exchange data using REST API between the Manufacturing Execution System and the aggregator 3.2. Protocol

Buffers are a data serialization format that is language- and platform-agnostic, making them well-suited for use in REST APIs. Protobufs are defined in a .proto file, which specifies the structure of the data. This makes it easy to transmit complex data structures over a network in a compact binary format, improving the performance and efficiency of the system, as exemplified on the testing made in [70], also shown Figure 3.3, in this study, the difference in time performance is minimal due to the small size of the transmitted message. However, as the message size increases, the cost of deserializing the message into JSON format increases correspondingly.

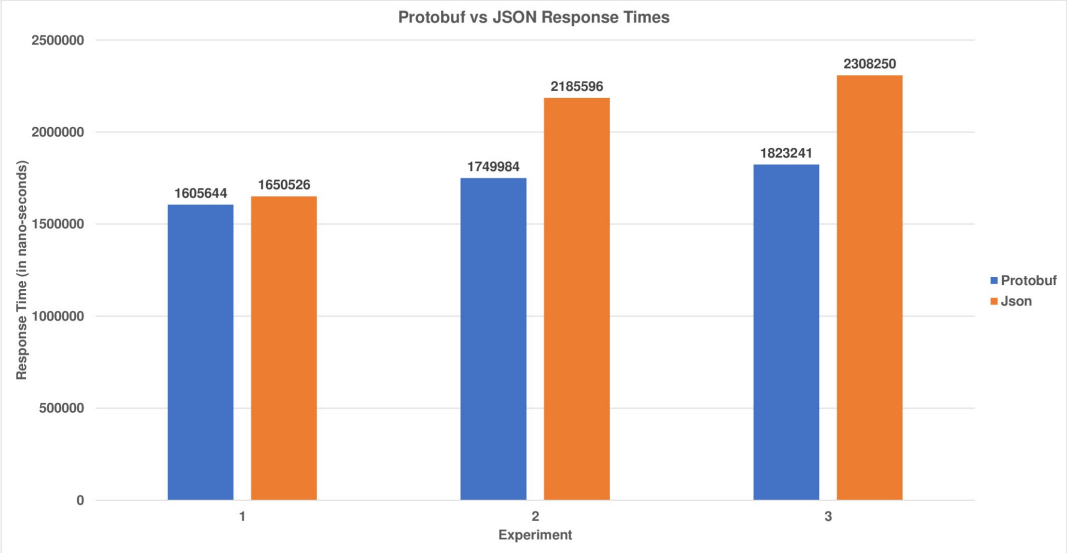


Figure 3.3: Response time comparison between JSON and Protobuf
Extracted from [70]

The illustration below, Figure 3.4 provides a visual representation of a portion of the contents contained within the .proto file used in the connection studied . The creation of this image was done using Protodot. It is important to emphasize that this graph only encompasses a fraction of the .proto file, as constructing a graph with all the implemented nodes and arrows would be impracticable. The aim of this graph was to exhibit a selection of the interconnections within the file, however, it is important to consider that the original file is significantly more extensive and intricate.

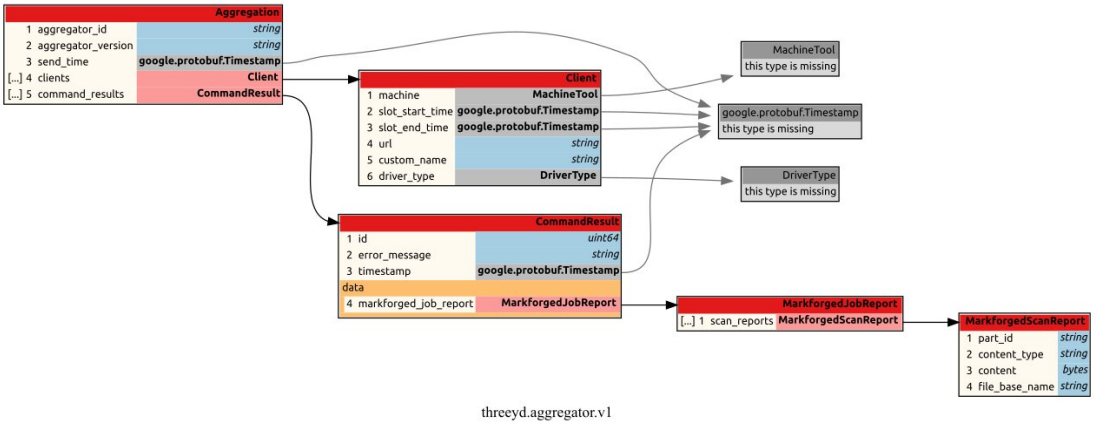


Figure 3.4: Schema of the 3YOURMIND aggregator communication

This implementation was put to the test using the company's tools and demonstrated satisfactory outcomes. It enabled the generation of a PDF document containing information on both the print process and communication data. The ability to initiate a machine and generate a comprehensive record of the print and communication data could enhance the overall project. Nevertheless, the trigger utility development was not pursued further in the project due to the restrictions imposed by industrial regulations and policies in Germany. These regulations and policies served as barriers that prevented the implementation of remote starting for the industrial machines, remote starting is still subject to the resolution of regulatory and policy restrictions.

3.1.2 Comparison between protocols

A reliable communication protocol is a key enabler of IIoT and therefore a crucial component for the effective development of Industry 4.0. If the IIoT solution is not robust enough, the data can be inconsistent, late, missing, or dangerously incorrect. For this research, the study conducted by Hottel in 2019 [71] and Wytrębowicz in 2021) [72] will serve as the foundational point of reference to compare the communication protocols reviewed for this implementation. The table below is a summary of the general differences between protocols

Table 3.1: Comunication protocols comparison

Protocol	OPC UA	MQTT	HTTP
Closed Firewalls	No	Yes	Yes
Low Bandwith - Low Latency	Partially	Yes	No
Real-Time communication	Yes	Partially	No
Interoperable data format	Yes	No	No

Both OPC UA and MQTT are standard communication protocols recommended by RAMI 4.0 [73]. RAMI 4.0 stands for Reference Architecture Model Industrie 4.0 and is a reference architecture developed by the Platform Industrie 4.0, which is a joint initiative by the German government, industry associations, and research institutions. Its reference architecture model provides a framework for structuring and organizing the different components and layers of an Industrie 4.0 system. While the RAMI 4.0 model is widely adopted in the industry and has become a de facto standard for Industrie 4.0, it is not a formal standard or official entity.

When it comes to to the comparison between HTTP and MQTT the latter has shown to be faster and more efficient, on 3G networks as the ones tested in this study [74], it is 93 times faster, uses 8 times less traffic, and consumes 170 times less energy on receivers. In other scenarios, such as this performance test [75], MQTT was at least 20 times faster, used 50 times less traffic, and was 22% more energy-efficient. A minor disadvantage of MQTT compared with HTTP is in the cost of connection setup [76]. As per the explanation in [77] the cases in which HTTP might be a preferable choice is to connect devices that already have an HTTP client installed. In [63] HTTP shown was found to have performance issues when implemented in constrained nodes. Constrained nodes are devices with limited computing resources such as processing power, memory, and bandwidth. Another comparable study [63] evaluated several protocols, namely RESTful HTTP, MQTT, CoAP, AMQP, DDS, and XMPP, and assessed their potential for implementation in

fog and cloud computing systems based on the Internet of Things. The researchers found that RESTful HTTP and MQTT were the most advanced protocols, with MQTT showing great performance on resource-constrained devices. One area in which MQTT sometimes falls short and could improve is security as explained in [78], solving some of the security issues could important impact on MQTT, specially when being compared with other available solutions. OPC UA, as explained in sections before [49],offers several advantages over MQTT, including secure and reliable data exchange with semantic interoperability. OPC UA allows for real-time data exchange while MQTT is not specifically designed for real-time data exchange. MQTT offers several advantages as communication protocol as seen above but is primarily designed for small-scale, low-bandwidth IoT devices and provides only a basic publish/subscribe messaging model while OPC UA provides a hierarchical structure for managing large amounts of data. It supports multiple data types, including complex structures and arrays, making it suitable for a wide range of industrial applications and for M2M industrial communication.

In summary, OPC UA is better suited for applications that require high reliability, security, and real-time data acquisition, while MQTT is better suited for applications that require low power consumption, low data rates, and low overhead. However, there may be some cases where both protocols can be used together to provide a more comprehensive solution, as explained in [79] and [80] The combination of OPC UA and MQTT is becoming popular in Industry 4.0 because it provides the benefits of both protocols and allows for interoperability between different industrial systems and devices.There is also studies like [66] that suggests that integrating OPC UA and REST can enhance interoperability and communication across different domains, thereby increasing the wider adoption of OPC UA as an authentic Internet-of-Things protocol.

3.1.3 Behavior Tree implementation

This section will present an in-depth examination of the implementation of the project's workflow. Firstly, the framework used to develop the project will be explained, highlighting the open-source tools and libraries employed, along with the reasons for their usage. Furthermore, the intended workflow of the entire system will be summarized and explained. As many nodes may have comparable functionalities across various machines, the emphasis will be on the underlying logic of the nodes. The communication process and the role of the blackboard in behavior trees will be examined, highlighting how information is retrieved from and transmitted to the machines and ultimately stored in the blackboard. Finally, the control nodes developed for this project and how they were implemented will be reviewed and discussed.

In this study, the C++ implementation of Behavior Trees employs the open source BehaviorTree.CPP library developed by Faconti and Colledanchise [35]. It is worth noting that the compatibility of Groot 1.0 is exclusively with BehaviorTree.CPP version 3.8.x, thus the demonstration presented in this paper, is constructed upon the framework provided by BehaviorTree.CPP 3.8.x.

The utilization of the BehaviorTree.cpp library for this project is motivated by several factors. Firstly, as discussed before, the library features a graphical user interface, Groot [36], for the visual description of workflows, which can then be loaded directly into the application, meaning if the nodes are already defined the behavior tree can run directly, in contrast to common state-machine libraries that require the manual translation of UML diagrams into C++ code.

In addition, it is worth noting that BTs are produced using XML, which enhances the readability of the trees, apart from the possibility of using the graphical interface. Moreover, the Groot library is capable of recording, replaying, and displaying the state of the workflow through the GUI. Furthermore, to facilitate the monitoring of nodes in behavior trees through Groot, the tool features a built-in monitoring functionality. This capability offers a visualization of the behavior tree's status during execution, facilitating the detection of potential issues and bottlenecks in the system. The panel provides a real-time visualization of the behavior tree's state, highlighting active nodes and those that are inactive. Additionally, the user can access any data associated with the nodes, such as variables or parameters, to assist with debugging and troubleshooting efforts.

Moreover, Groot allows to log data from the behavior tree to a file or to a database, the log replay feature enables to replay the execution of the behavior trees from a log file generated during a previous run, thereby facilitating debugging and comprehension of large trees, like the one in this study. To utilize the log replay feature in Groot, a log file needs to be generated during the behavior tree

execution, which can be achieved by setting up a logger for the behavior tree and configuring it to write log messages to a file. Subsequently, the log file can be loaded into Groot and used to replay the execution of the behavior tree. The picture below, in Figure 3.5, displays an illustration of the log replay for a Failure scenario for a particular subtree. The practical application of this capacity has proven crucial in real-world testing scenarios. The ability to execute and monitor the behavior tree in real-time facilitates rapid iteration of the tree's design, enabling expeditious modifications to the tree's structure. The feature to execute and monitor the behavior tree in real-time expedites the iteration of the tree's design, facilitating prompt modifications to the tree's structure. This functionality can be particularly advantageous in situations where a partner may request a specific method invocation, necessitate the repetition of a particular workflow, or require alterations to the current workflow design.

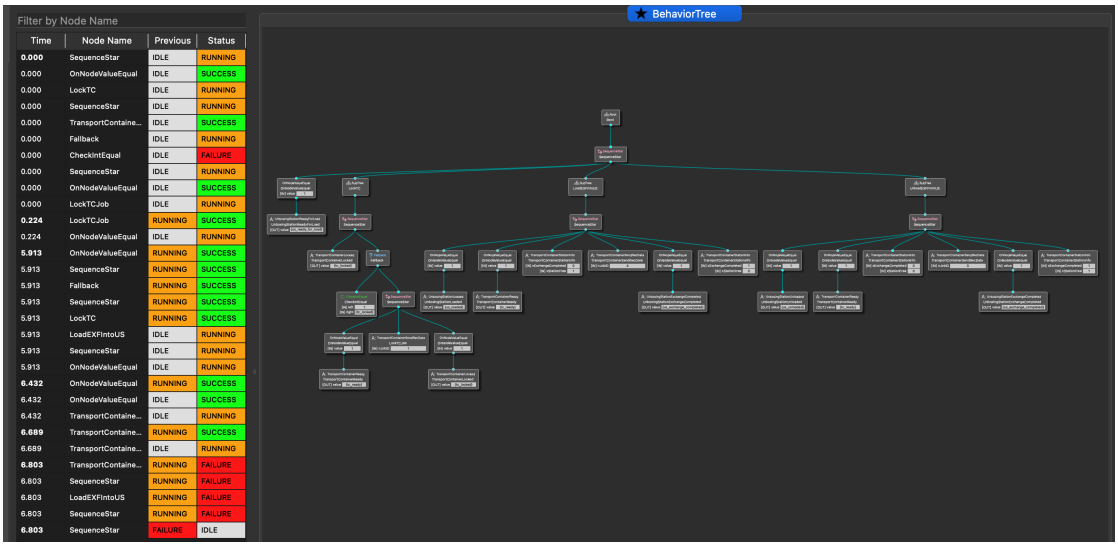


Figure 3.5: Log replay from the testing of one subtree.

Lastly, it can be highlighted that the dynamic loading of the Behavior Tree structure at runtime allows for on-the-fly adjustments to the system's behavior, eliminating the need for program recompilation. Dynamic loading of the Behavior Tree structure at runtime refers to the ability to modify the structure of a behavior tree while the program is running. This feature allows to change the behavior of a system without having to recompile and redeploy the entire program. This feature is especially advantageous when testing the system's workflow with physical machines in real-world environments, as the one described in this research. As it

will be discussed in the last section 3.6 this flexibility has enabled to create more robust and adaptable systems, as it makes possible to easily adjust and fine-tune the system's behavior to meet specific requirements or handle unexpected situations, in real world testing.

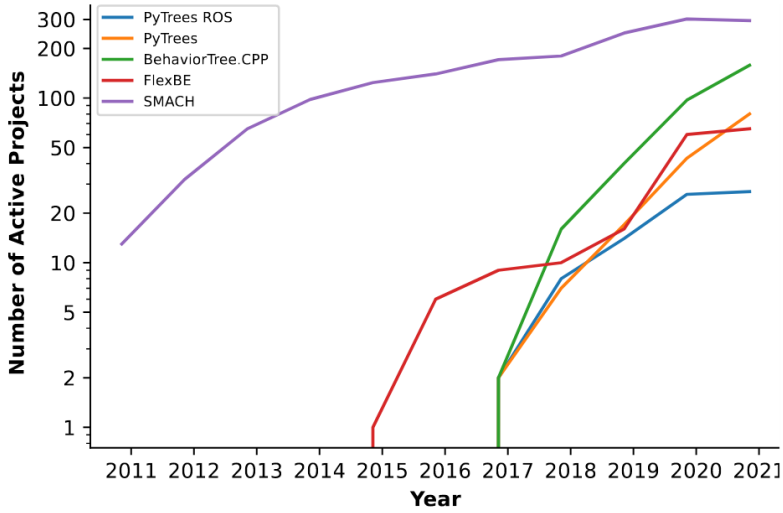


Figure 3.6: A study of the trend of usage of state machine and behavior tree languages in open-source projects on GitHub over time.
Extracted from [31]

Moreover, as illustrated in Figure 3.6 and expounded upon by Ghzouli et al. (2022) [31], the BehaviorTree.cpp library has recently gained substantial traction and adoption in the field. This trend can be attributed to the library's capacity to efficiently model and regulate the behavior of autonomous agents by means of a hierarchical tree structure, as well as its provision of numerous exemplary cases that facilitate the establishment of research settings.

3.1.3.1 Workflow design

In the subsequent section, a detailed examination of the workflow's design and corresponding machine actions will be presented. The following sub-sections will provide a comprehensive outline of the distinct phases in the workflow and scrutinize the specific tasks undertaken by each machine.

The image depicted on Figure 3.7 provides a comprehensive representation of the spatial arrangements of the machines and their respective interactions. Positioned on the left side of the image are the EOS P500 3D printer, the Grenzebach Transfer Station, and the Grenzebach Fleet Management System, controlling the automated guided vehicles (AGVs) that transport the transport container. After the printing process is completed, the printed file is affixed onto an exchange frame (EXF), which is subsequently moved onto the Krumm-tec Unpacking Station, located in the middle. Situated on the right-hand side of the system configuration, the Falcon DyeMansion (sandblasting and cleaning machine) is responsible of the post-processing of the printed part. The Grenzebach RobotCell, which encompasses the handling and bin-picking components, enabling the transportation and clasification of boxes within its confined space. The boxes that carries the printed parts in the different stages of the workflow are of two types - large and small, with the handling robot capable of transporting both. In contrast, the bin-picking component can only shift items into small boxes situated near its location. Finally, the workflow culminates at the entrance of the system, where individuals may retrieve the finished printed parts, as a conclusion of the manufacturing process.

As of the current state of the project, the DyeMansion coloring machine and the Nabertherm oven are not yet operational. The delay in their utilization is primarily due to the lack of the module that facilitates the automatic loading of the machine, denoted as AGV position on the images. As such, the AGvs cannot load the machines automatically. Currently, the part heating process is occurring directly on the printer.

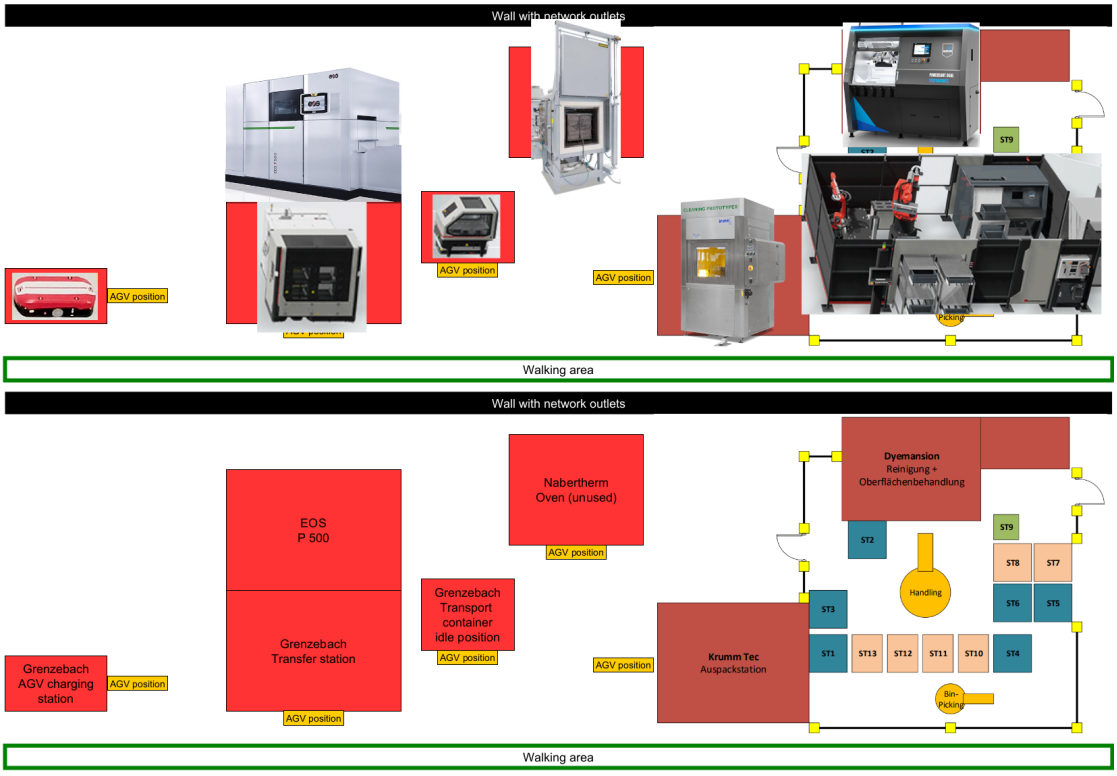


Figure 3.7: Workflow schema with machine representation
Source 3YOURMIND

3.1.3.2 Initial development

Prior to delving into the specific details of the processes involved, an exposition of the logic steps employed to create the decision trees will be presented. The decision tree was originally conceived with the aim of scrutinizing each machine process in isolation, mainly due to the ease of interpretation and signaling processing that this option offered. To illustrate this point, the photograph below showcases the Nabertherm oven as an example. It is noteworthy that during the design and testing phase of this research, the Nabertherm schema and signals were established as the initial and revolving point of logic for this investigation, since the signals for this machine only involve, the initiation or termination of the process, as well as the opening or closing of the door. In this idea of thought the first trees created were the ones to open the Nabertherm Door, shown in Figure 3.8, close the Nabertherm Door, and Start and Stop the heating, they originally would have become subtrees of the final tree.

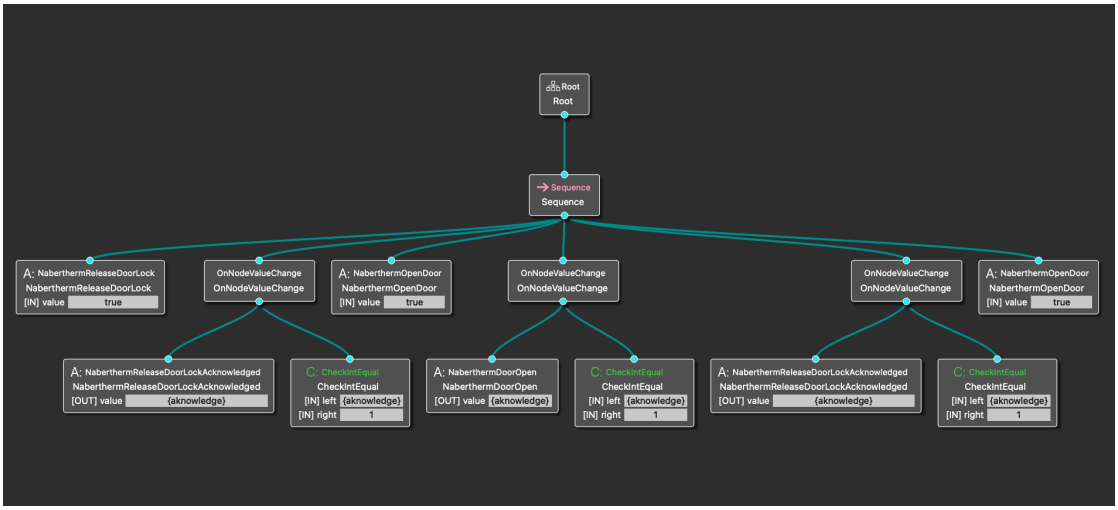


Figure 3.8: First schema of the Nabertherm OpenDoor Behavior Tree

After further testing this option, it became evident that this approach was not the best course of action. The lack of representation of the collaboration between processes and machines in the plant compromised the overall effectiveness of the decision tree. It was determined that the approach needed to be altered to incorporate a broader perspective that incorporated the collaboration between different processes and machines, the solution presented was to construct the final decision tree based on specific operations that involve the integration of the different machines from the project. The integration of multiple processes and machines allowed for a better understanding of the relationships between the different operations and their impact on the overall outcome and as it will be discussed in Section 3.1.3.9, this approach enable to initially test the whole simulation viability in a mock environment.

After this testing the required workflow was implemented and divided in different sections. The subsequent three sections, that can be seen in Figure3.9 outline the number of programmed days of the original testing. It is important to note that the division into days is merely for the purpose of organization and does not necessarily indicate that each of the three sections will take a full day to complete.

The workflow has been divided into three distinct days, with each day comprising of sub-sequences that can be completed within one day. On the first day, three partners are involved, namely the EOS P500 3D printer, the Grenzebach Transfer Station, and the Grenzebach Fleet Management System (AGVs). The second day involves the Grenzebach Transfer Station and the Krumm-tec Unpacking Station, while the third day entails the Krumm-tec Unpacking Station, the Falcon DyeMansion, and the Grenzebach RobotCell.

One of the key features of Behavior Trees is that they are executed sequentially, meaning that each node in the tree is visited in a particular order, typically from top to bottom. Due to its sheer size, the final behavior tree file [tree.xml] is divided into subtrees that are executed sequentially, this division is primarily for the purpose of enhancing the file's readability and facilitating its creation and debugging. The main behavior tree is compartmentalized into four distinct sections, each with a specific purpose based on different parts of the workflow (printing, post-processing, robot selection,...). As seen in the picture below, Figure 3.9, this four-sectioned structure constitutes the primary behavior tree and represents the main logic behind the workflow.

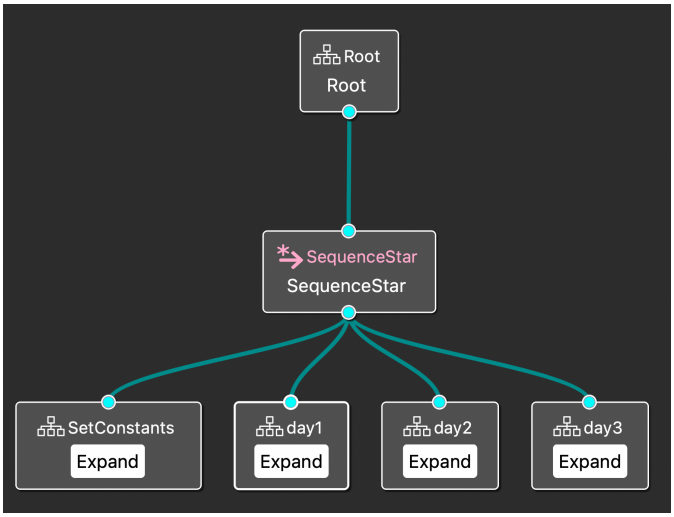


Figure 3.9: Main Behavior Tree of the project

The initiation of the workflow involves the configuration of necessary parameters for the complete workflow. The initial step in the tree employs the utilization of a node referred to as the SetBlackboard node, which is already available in the BehaviorTree.cpp library. This node serves the purpose of defining the values of the signals utilized for communication between the tree and the machines. The initial phase of the workflow is encapsulated within a subtree referred to as "SetConstraints." This subtree serves as a foundational step to establish the parameters required for the overall workflow.

The SetBlackboard node is a type of Action node in Behavior Trees that is used to set a value on the Blackboard. As explained before, the Blackboard is a data structure that is used to share information between different parts of the Behavior Tree. Upon execution, the SetBlackboard node takes in a key-value pair and sets

the corresponding value on the Blackboard. The key represents a specific variable, while the value can be any type of data, such as a number, string, or object.

It is important to note that this initial step in the workflow will be improved and refined in the future by a graphical user interface (GUI) that goes beyond the scope of this thesis

The division of the workflow into different days made the designing process more manageable. Now a summary of the action and processes involved in each day will be presented.

The first-day processes involve the collaboration of three different partners and machines, which include the EOS P500 3D printer, the Grenzebach Transfer Station, and the Grenzebach Fleet Management System (AGVs). The second day of the workflow is focused on the Grenzebach Transfer Station and the Krumm-tec Unpacking Station and there is participation again from the Grenzebach Fleet Management System. The third day of the workflow involves the Krumm-tec Unpacking Station, the Falcon DyeMansion, and the Grenzebach RobotCell. A graphical representation of the workflow setup can be seen in 3.11.

It is important to highlight as well that within the transfer station and all of the other machines, there exists a conversion point where the Transport Container (TC) is placed by an Automated Guided Vehicle (AGV) under the control of the Fleet Management System. The aforementioned transportation container has the capability to transport the exchange frame, where the part is printed, shown in Figure 3.7.

3.1.3.3 Workflow day 1

The initial condition on day 1 is that the printer exchange frame (EXF), where the parts are printed, is located within the InOutFeed section of the transfer station (TS), blue square on the schema from 3.2. At the conclusion of the day, the workflow controller should have issued all the required commands to bring the process to the final state, where the EXF with the printed parts is positioned inside the printer, located within the TS.

On day 1 of the diagram, the workflow checks if the mentioned prerequisites are met and if the serialization of the Additive Marking is done. It is important to note that the serial marking check node has not yet been implemented as it is presently non-functional in the on-site project. After checking the preconditions, the workflow triggers the OPC UA commands and nodes that required for the TS to relocate the empty EXF from the InOutFeed to the 3D printer, meaning the progression from X1Y1Z1 to X2Y1Z1, better depicted in Figure 3.10. After the EXF has been relocated, the printing process may commence. However, for legal reasons in

Germany, the initial phase of the printing process must be manually activated by an operator pressing a button, as it is not permissible to start industrial machines automatically. The 3D printer then performs the required tasks automatically, and then the exchange frame, which is positioned already in the designated location carrying the printed component, will undergo a cooling process.

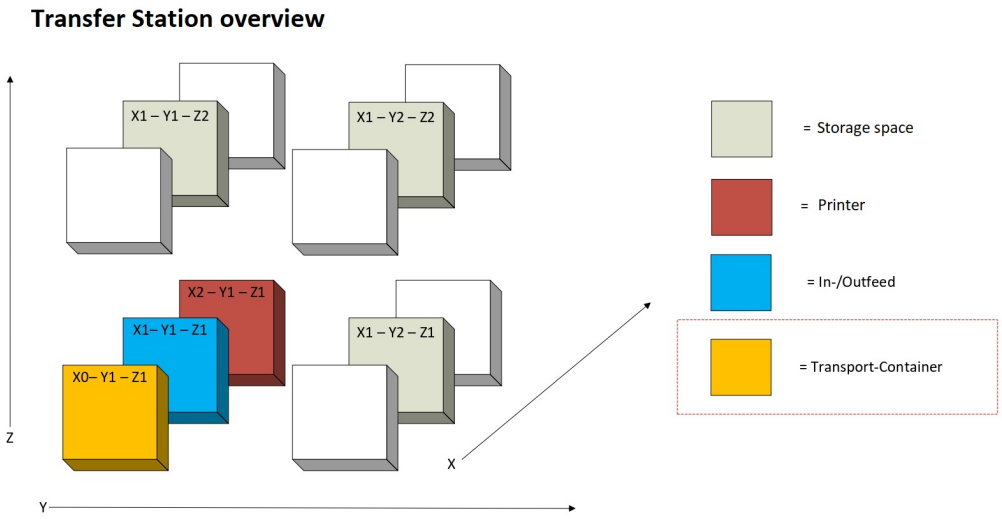


Figure 3.10: Transfer Station logic overview
©Grenzebach (Edited and translated from German)

3.1.3.4 Workflow day 2

On the second day of the workflow the general tasks are the following: the printed part is within the printer and cooled down, and is transferred from the EOS P500 3D printer to the transport container, shown as the yellow square in 3.10. The transport container is then moved by an AGV to the Krumm-tec Unpacking Station for further processing. The Day 2 workflow depends on specific preconditions, namely, that the printer component has been deposited within the printer and subsequently cooled down, and that the transport container, which is transported by AGVs, has been secured in the locking position denoted as X0Y1Z1 in Figure 3.10. Upon verifying the aforementioned conditions, the exchange frame is relocated to the InOutFeed station, meaning a transition from the red box to the blue box in Figure 3.10. It is worth mentioning that any exchange frame intending to exit

the transfer station must first pass through the InOutFeed station. Upon the relocation of the exchange frame to the InOutFeed station, the workflow sends a command to the Transport Container, instructing it to secure itself into the appropriate position for transfer. Subsequently, a directive is issued to open the transfer station door. Once the transfer station door is opened, a command is dispatched to the Transport Container, ordering to retrieve the Exchange Frame from the Transfer Station. Given the lack of communication between the Transfer Station and the Transport Container, the Behavior Tree, in conjunction with OPCua communication,is responsible for notifying both parties of any relevant changes. Thus, the workflow controller issues a message to the Transfer Station, indicating that the InOutFeed station is now empty, followed by a signal to close the transfer station door.

Upon completion, a message is sent to the Transport Container to unlock itself, at this point the fleet management system is notified to dispatch one of its vehicles (AGVs) to retrieve the Transport Container and carry it to the Krummtec Unpacking Station (US). The fleet management system handles the required vehicular movements, and only requires notifications indicating the source and destination of the Transport Container. Upon reaching the unboxing station, the fleet management system informs the workflow controller that the Exchange Frame has been deposited. The process is reiterated in the Unboxing Station, wherein the Transport Container is once again locked into position, the Exchange Frame is unloaded and subsequently conveyed to the unboxing station along with the printed components. Once the exchange has been completed, the Transport Container is unlocked and directed to return to the Transfer Station, where it will proceed to initiate a new cycle. Meanwhile, the unboxing station initiates the unpacking procedure, following which the unpacked component is relocated to Station1, shown in Figure 3.7, which is designated as the location for the installation of the printed component. The Unboxing Station contains the Large Empty Dirty (LED) box, which serves as the designated storage location for the printed component when is unprocessed. The result of the printing process is a large box full of powder with the printed part inside the task of the Unpacking Station is to isolate the printed component. After the process, the box with powder residue is designated as Large Full Dirty (LFD) box.

3.1.3.5 Workflow day 3

The workflow on Day 3 has the prerequisite of having the LFD box and its constituent parts located in Station 1. The objective of this last part of the process is to have all of the individual components classified and prepared for removal from the system by the end of the day. This is to ensure that the workflow is successfully concluded (in Station 7) in Figure 3.7 and that the components are readily available for use outside the system.

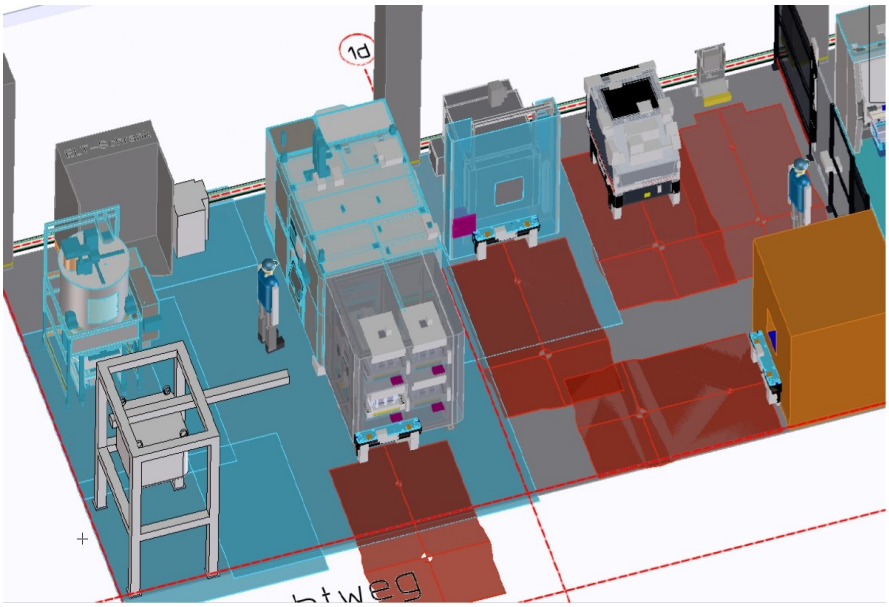


Figure 3.11: Graphical representation of the workflow setup ©Grenzebach

The Day3 subtree starts by directing the relocation of the printed component box to the DyeMansion Falcon post-processing machine and ask the Grenzebach Robot Cell to move from Station1 to Station 2.1 (which is utilized for accommodating full dirty boxes at the Falcon, and will store the cleaned parts at the conclusion of the post-processing). In addition, an empty large box is relocated to the Falcon and positioned at 2.1, serving as the recipient for the post-processed cleaned parts. The transfer of dirty boxes is limited to Station 1 from US to Station 1 of Falcon to ensure that there is no confusion between the clean and dirty boxes. Clean boxes are initially sourced from Station 3, which serves as a buffer, their availability is a prerequisite of the workflow. Although it is possible for clean boxes to be fed into

Stations 5 and 6 by personnel, this is not part of the workflow considerations, and it is assumed that there will always be at least one clean box in the buffer in Station 3. The Behavior Tree commands the Robot Cell to transport the clean boxes from the buffer to the Falcon, which then proceeds to undertake all three steps of the process in a single operation. This entails loading the parts from the box, cleaning them, and subsequently repositioning the empty dirty box and loading the large clean box. Once the Falcon has completed the task, the robot cell is instructed to return the empty dirty box to the unboxing station. Upon completion of the previous steps, the clean box containing the clean parts is relocated to Station 4, which is designated for bin picking. At this station, the parts are scanned and separated based on their classification, this station has the capacity to detect up to four distinct parts and allocate them into the relevant sub-stations (St10 - St13), which must already contain small empty clean boxes to accommodate the individual components. Prior to the allocation process, the robot must be informed of the expected number of parts, as it is the responsibility of the system operators to determine the specifications of the printing process. Accordingly, the operators must inform the robot of the number of identical parts that will be produced in a print run, as well as the number of distinct parts that will be generated. Subsequently, the bin-picking operation is performed. The empty large clean box is then transported from Station 4 to the buffer for utilization in the next cycle. Next, the small boxes are conveyed to the outfeed where individuals can retrieve them.

However, the additional precondition specifies that space must be available at the outfeed to accommodate all of the boxes and that they must be retrieved in the appropriate sequence (i.e., one at a time). Upon completion of the first loop, new clean and empty boxes must be transported to the system, which is designated as the second loop of Day 3. Depending on the number of parts printed, there may be up to four boxes required. Thus, the full boxes with the parts are transferred from Stations 10-13 to Station 8, while empty boxes are relocated from Station 7 back to Stations 10-13.

3.1.3.6 Machines and Behavior Tree Nodes

This section presents a detailed overview of all the Leaf (action) nodes that were implemented by 3YOURMIND for the project and how they were interconnected with the manufacturers and machines involved in the workflow.

Table 3.2: Manufacturers, machines and BT action nodes from the project

Manufacturer	**Machine**	**Nodes**
DyeMansion	Falcon	DMFalconAbortJob DMFalconLoadStateId DMFalconReady DMFalconSetJob DMFalconStart DMFalconUnloadStateId
Krumm-tec	Unpacking Station	UnboxingStationBusy UnboxingStationError UnboxingStationExchangeCompleted UnboxingStationLoaded UnboxingStationReady UnboxingStationReadyForLoad UnboxingStationForUnload UnboxingStationUnloaded
EOS	EOS P500	EosExchangeFrameLoadCommandState EosExchangeFrameUnloadCommandExecute EosEschangeFrameUnloadCommandState EosJobStartCommandExecute EosJobStartCommandState EosPauseCommandExecute
Nabertherm	Oven	NabertherCloseDoor NabertherDoorClosed NabertherDoorOpen NabertherOpenDoor NabertherReleaseDoorLock NabertherStartProgram NabertherStopProgram
Grenzebach		FleetManagerGetTaskStatus FleetManagerTriggerPickAndDrop

Manufacturer	Machine	Nodes
	Transport Container	TransportContainerBusy TransportContainerExchangeCompleted TransportContainerLoaded TransportContainerLocked TransportContainerReady TransportContSendRecDta TransportContStatonInfo TransportContUnloaded TransportContUnlocked
	Transfer Station	TransferStationBufferEXFReady TransferStationDoorOpen TransferStationExchangeCompleted TransferStationInOutFeedEXFReady TransferStationInOutFeedEmptyEXF TransferStationInOutFeedFree TransferStationJob TransferStationOperateDoor TransferStationPrinterEXFReady TransferStationPrinterFree TransferStationREadyForJob TransferSTationStatusInOutFeed
	Robot Cell	RobotCellSendOrder RobotCellActualPartsAtSation12 RobotCellBinPickingRobotReady RobotCellChangeProcessState RobotCellHandlingRobotReady RobotCellHandlingState RobotCellInFeedSmallKLTInPos RobotCellOutFeedSmallKLTInPos RobotCellSendSetpoitsForBinPicking

3.1.3.7 Node communication

In order to comprehend the underlying logic behind the implementation of the Behavior Trees and its actual functionality, it is imperative to explain the logic behind the communication of the nodes with the machines involved in the workflow. This section will examine the underlying principles governing the communication between the nodes and machinery integrated into the workflow. It is important to note that the connections discussed in this study are made based on the OPC UA communication protocol, which was selected and deployed on the machines by the manufacturers involved in the workflow. The OPC UA communication implementation with MES was developed by 3YOURMIND prior to the current study, and serves as the foundational framework for the connections established in this project.

In this context the key piece of functionality is the Blackboard, as explained in Section 2.2.1.2, a blackboard is a shared data structure that allows nodes within a Behavior Tree to communicate and exchange information with each other. It serves as a centralized storage location for variables and data that can be accessed by multiple nodes. This allows the Behavior Tree to make informed decisions based on the collective information available on the blackboard. BehaviorTree.CPP implements a flexible, user-friendly port-based system to facilitate data flow between nodes. In this context, a Blackboard serves as a simple key/value storage mechanism that is shared among all the nodes in the tree.

Input ports are capable of accessing information stored in entries in the Blackboard, whereas output ports can create new entries in the Blackboard. As a result, output data from one node can be used as input for another node in the behavior tree.

Within the scope of this project, a node within the Behavior Tree can assume several roles, including representing an OPCua data point associated with a machine (e.g., boolean, bit, or integer), an OPCua method used to interact with the machine, a control node such as sequence or fallback, a condition node, or an entire subtree.

To explain the nodes is vital to explain the connection between the machines, In the previous sections, the OPC UA connection logic was discussed. Even though its development was not directly part of the research process, it is crucial to explain it for a better understanding of the node's logic. Machine Connectivity plays a critical role in storing the values of the OPCUA server nodes associated with each machine. For instance, when an output behavior tree node is ticked, the value of the OPCUA node from the server is recorded and saved onto the blackboard.

Specifically, when a designated output behavior tree node is ticked, it copies the current value of the node in a thread-safe manner and sets it into the output port, which is typically remapped to a blackboard key using the syntax.

To explain this logic better it is possible to take the XML syntax below from the tree. When this node `TransferStationPrinterFree` is ticked

```
1 <Action ID="TransferStationPrinterFree" value="{
    ts_ready_printer_free}"/>
```

the key `"{ts_ready_printer_free}"` saves the value of the OPC UA node represented by `"value_nodes.transfer_station_printer_free"`, onto the Blackboard. In this case, the `value_nodes.transfer_station_printer_free` saves the OPCua value that shows the status of the printer.

Conversely, in cases where an input value is ticked within the Behavior Tree node, the code extracts the value from the input port, remapped to a blackboard entry, and executes an asynchronous action based on its value.

Regarding Grezenburg, for example, the manufacturer of the Transfer Station, the OPCUA nodes utilized in this study were developed from the Siemens OPCua Modeling Editor. This tool is used to develop and edit the information models that describe the nodes and variables used in OPCUA-based applications. Typically, the signal received in the majority of cases takes the form of a Boolean signal. Input signals representation can be seen in 3.13, better explained in table3.4. Output signals examples are displayed in 3.12, better explained in 3.3.

The tables and pictures below showcase the implementation explained

Table 3.3: Example of a blackboard key

BLACKBOARD		
Key	Type	Value Node (from OPC UA server)
ts_ready_for_job	Boolean	transfer_station_ready_for_job

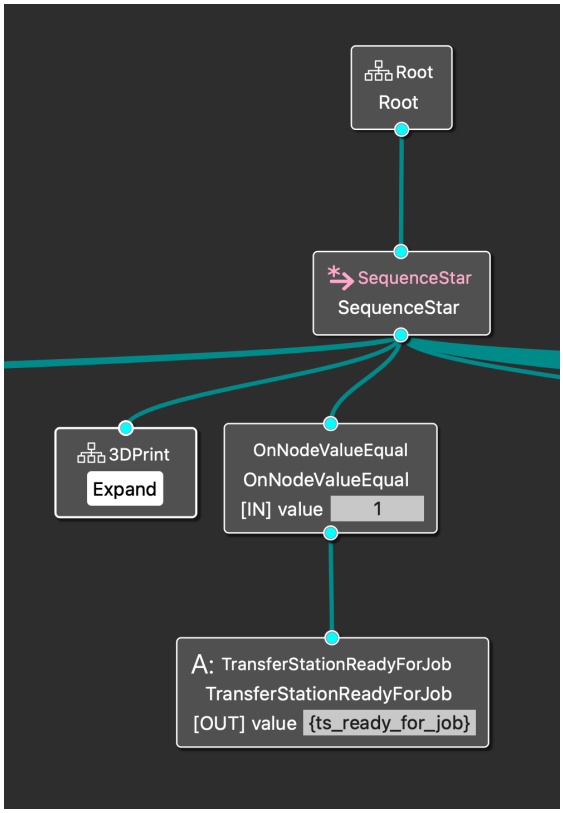


Figure 3.12: Example of output nodes

Table 3.4: Example of a blackboard key

BLACKBOARD		
Key	Type	Value Node (Writes on the OPC UA server)
open_door	Boolean	nabertherm_open_door

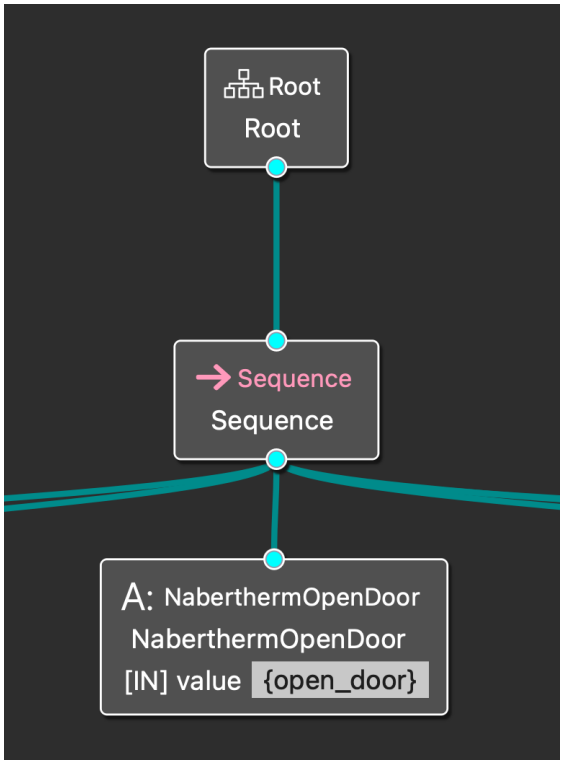


Figure 3.13: Subtree representation, example of input nodes

3.1.3.8 Control nodes

In this final section, the control nodes that were integrated into the Behavior Tree framework used in the workflow will be examined. Control nodes are a type of node in Behavior Trees that play a vital role in the flow of execution within the tree. They serve to organize and group other nodes within the tree, enabling the direction of the behavior. The control nodes utilized in this workflow include both traditional nodes, such as Sequence and Fallback, and less common control nodes like SequenceStar and OnNodeValueEqual. These specialized nodes were selected for their unique ability to effectively address the specific challenges in this production line. Apart from the common nodes already explained in Section **??**, we have to also highlight some of the nodes from the BehaviorTree.cpp library most used, in them we fin the SequenceStar node, seen in Figure 3.9 and 3.12 for example. The Sequence and SequenceStar nodes are both used in behavior trees to execute child nodes in a specific order, but they have different behavior. The Sequence node starts executing its child nodes from the beginning each time it is ticked, and stops executing as soon as any child node fails. On the other hand, the

SequenceStar node starts executing from the last failed child node, and continues executing all remaining child nodes. The SequenceStar node has been renamed to SequenceWithMemory in the latest version of the BehaviorTree.cpp library. This nodes with memory are useful for cases where it is know that a node doesn't have to be executed after the first time. Memory nodes store the success or failure of a child node and the memory is reset when the parent node returns success or failure, allowing all child nodes to be considered during the next activation. It's important to note that control flow nodes can still be executed without memory nodes by using auxiliary conditions, making memory nodes a convenience.

OnNodeValueEqual node is used to compare the value of a child node against a predefined value. The node inspects the status of the child node and, in the event of success, records the child node's value in a variable named `"child_value."` The node then evaluates whether the stored value is equivalent to the specified value, and returns a success status, or a failure status otherwise. This node is widely utilized in this implementation, as exemplified by its prevalence in figures like 3.12. Specifically, it has been mainly used to verify that a given OPC UA node has returned a Boolean variable with a the desired result.

3.1.3.9 Mock environment results

The complete POLYLINE project has been integrated and tested within the BMW production structures to verify its functionality and performance within a real-world production environment, however, since testing within BMW's facilities necessitates the collaboration of numerous companies, it was forecasted that it may not be feasible to complete the testing prior to the submission of this research. Hence, simulation was deemed as the primary testing tool to evaluate this research viability and potential for implementation. The aim is to ensure that the developed implementations are capable of meeting the workflow objectives, thorough testing and simulation has been carried out. The tests have been focused on the well development and safety of the workflow and the ability of the system to support the workload generated by the production process. The results have shown that the workflow controller is able to support the workload generated by the production process, ensuring a smooth transition between the different stages of the production line.

In this research, the software development methodology adopted is Test-Driven Development through automated mock testing using the GoogleTest[1] frameworks in C++. This approach has become widely popular in the software engineering community, due to its demonstrated efficacy and effectiveness over time. TDD

[1] http://google.github.io/googletest/

emphasizes writing automated tests before writing actual code, which serves as a guide for development and ensures that the code meets the desired specifications. As a result of its widespread success and proven track record, TDD has now become the de-facto standard in software development.

Google Test provides a framework for writing and running automated tests for C++. In gtest, automated mock testing can be performed by using mock objects to simulate the behavior of dependent components in the system under test. In the context of this research, the functionality of nodes can be tested without connecting to actual machines in the real worflow, and instead simulating the values received from the OPC UA server.

As part of the testing process, various scenarios were evaluated to ensure the functionality and reliability of the system being developed. This involved the evaluation of situations that required the leaf nodes to communicate failure status to the root node, such as the door of the transfer station not being able to open, the unboxing station not being ready, and the 3D printing not being completed. These scenarios were selected as they represent common failure conditions that may arise in the system and provide insight into the system's ability to properly respond to and recover from such situations. It is important to emphasize that the most important scenario to undergo testing has been the "happy path" scenario, a widely recognized concept in the realm of design and software, in general, [81], in which all connections operate efficiently within the allotted time frame and the tree, as a whole, returns a value indicating successful completion following the callback.

To accomplish this, mock objects of the nodes were created. These mock objects simulate the signal reception from the OPC UA nodes through the blackboard keys between the nodes and the real machines servers in the system, allowing for testing without the need to connect to the physical machines.

This approach to testing provided several benefits as it allowed to confirm that the implementation made was capable of handling the workload generated by the production line. Futhermore this approacche made it easier to identify and isolate problems in the system. In conclusion, GoogleTest provided an efficient and adaptable option for testing ensuring the validity and reliability of the developed implementations for the successful integration of the POLYLINE project within BMW's production structures.

3.1.3.10 On-site results analysis

Although the workflow controller is still undergoing testing, preliminary analysis of the data has been possible by extracting information from the log files generated by the machines in the first 10 days of testing. This section aims to show the preliminary discoveries and findings made during the initial testing stages of the project. Furthermore, it serves to verify the theories proposed before the start of the project's implementation, particularly those related to Behavior Trees, and highlight areas that require further improvement. It should be noted that these findings are from the testing phase and may vary in the final outcome, however, they still provide crucial information for future development.

A significant outcome from the on-site testing was the recognition of the advantages offered by Behavior Trees for this project, specially in terms of modularity, scalability, and reusability. These attributes proved to be especially useful as the real-world environment in the factory was still undergoing development and not all machines could be tested since the beginning. As a result, multiple revisions needed to be made to the final Behavior Tree file `[tree.xml]` to incorporate different sequences of production. The hierarchical nature of Behavior Trees facilitated the isolation and resolution of issues, by easily adding or removing processes using subtrees. This case study demonstrates the advantages of using Behavior Trees in designing complex systems that require challenging decision-making, as they can be easily adjusted and fine-tuned to meet specific requirements and handle unexpected situations.

At present, the ability to automatically analyze the data from the production line has not yet been implemented.. Nevertheless, a Python code [2] was developed to extract meaningful insights from the log data and present them in a readable format. The log files contained detailed information about each of the events that occurred during the testing phase, including time stamps, errors, or issues that arised.

The objective of this analysis was to identify the key performance indicators, such as the average duration of a manufacturing cycle or the average interval between error outputs, in the same way, the analysis will aim to locate the source of errors during cycles, all of that could be used to reveal potential areas for improvement.

[2] https://github.com/zaidabri/DataAnalyticsFromLogFiles

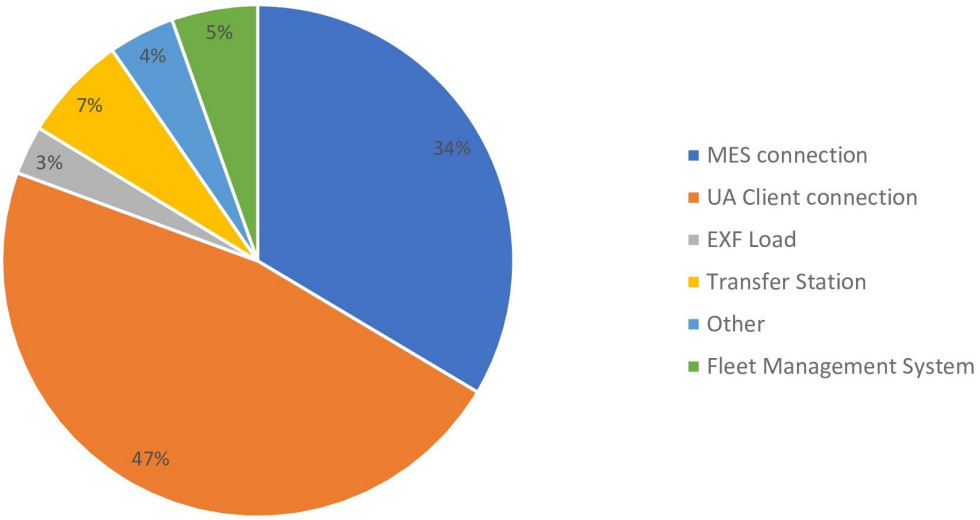


Figure 3.14: Schema of the main source of errors in the testing

The results throw that two successful entire prints have been made with an average duration of ≈ 19 hours, which is promising since the printing itself lasts an average of six hours and the cooling process is planned to last double of the printing time, shown in Figure 3.14. However, only designated subtrees have been subjected to testing so far, with the full tree remaining untested.

It is notable as well that the primary source of error on the aggregator is maintaining the connection with OPC UA, followed by the communication with the Agiles MES. On average there is an error message returned by the aggregator every ≈ 35 minutes.

Transferring the Exchange frame from the InOutFeed on the Transfer Station to the Transport Container 3.10 is identified as the most frequent source of failure in the workflow execution, this could be attributed as well to the fact that this particular step has undergone more extensive testing. The next source of errors is related to the Fleet Management System. On average there is less than 1% of Failure return every cycle.

In summary, this initial analysis establishes the baseline value for the KPIs designated in this research. This value results throw information in the areas where to improve furthermore, and confirm that there is potential for automation in additive manufacturing

Chapter 4

Conclusion

Additive manufacturing has the potential to revolutionize the manufacturing industry by providing a multitude of benefits like accelerated prototyping, exceptional adaptability, and heightened design flexibility. However, the automation of additive manufacturing processes has been challenging due to the complex processes and decision-making steps involved in operating an AM production line. The proposed controller in this research offers a systematic and effective way to manage the complex processes involved in the designated workflow.

The research in this thesis provides valuable insights into the potential of automation in additive manufacturing, it addresses and analyses the possible advantages that Behavior Trees offer over other similar methods like Finite State Machines or Hierarchical Finite State Machines. Both simulation and on-site testing have highlighted the advantages of using Behavior Trees, especially in terms of modularity, scalability, and reusability. As the final workflow had to be modified several times to adapt to changes in the real machines or in the steps in the workflow, the ability to create and reuse subtrees to form larger systems and add or remove certain processes from the workflow has been a great help in the successful development of the implementation.

When it comes to the libraries employed, the framework provided by *BehaviorTree.cpp* has been decisive since it has allowed to easily create the personalized nodes for the project and also to efficiently implement the workflow logic through the use of pre-existing nodes in the library. The main nodes provided from this library that can be highlighted are the common control nodes as *Sequence* and *Fallback*, the control nodes with memory like *SequenceStar* and lastly the action node *SetBlackboard* use to set up the inital constants of the workflow. Apart from that, the personalized Leaf action nodes in general were in change of gathering and sending data from the aggregator to the OPC UA server from the machines. The possibility of creating personalized control nodes was crucial was for the develop-

ment of the general workflow, for instance, the *OnNodeValueEqual* node enabled the comparison of the node's return value against a specific value, resulting in improved efficiency by eliminating the need to create particular condition nodes that analyzed the particular value of a signal.

In the same way, the graphical editor provided by *Groot* has proven to be an excellent tool, particularly for the collaboration between teams as it allows for graphically and rapidly edit parts of the tree. Furthermore, the possibility to record and replay the workflow has been very beneficial to further understand the system response when testing. Additionally the possibility of editing the trees in real-time without having to recompile the tree, that has been a really helpfull for in-site testing.

The conclusions in this study show that BTs are an effective model to automate AM manufacturing systems. The flexibility of behavior trees to dynamically make decisions during run-time and incorporate new equipment into the system through nodes is very advantageous. With the used of BTs for cloud workflow controllers it is possible to adapt to rapidly changing situations within the manufacturing environment, thus enabling operators to quickly modify decision-making trees as required. This makes it easier to integrate new machinery into the system without undergoing significant modifications. The blackboard architecture offers a flexible framework for integrating data from various systems, such as the machines from different manufacturers used in this project. This enables distributed problem-solving by allowing independent machines to access and modify the common knowledge store of the tree.

The research findings indicate as well that testing using simulation, in particular through the creation of mock environments is a viable way to proceed. The workflow controller was primarily tested in a mock environment and the results demonstrated that it was capable of supporting the workload generated by the production process, ensuring a smooth transition between the different steps in the production line.

This research conducted a comprehensive study of the principal communication protocols used in the Industrial Internet of Things to provide a more comprehensive assessment of the proposed connections for the successful implementation of the workflow and gain a comprehensive understanding of the underlying automation functionality intrinsic to the project. The primary means of machine communication used in the project was based on the OPC UA protocol. Furthermore, as part of this research, tests were carried out on machines that utilize the REST API as a mode of communication to study the possibility of creating a trigger utility for workflow in the project. Additionally MQTT protocol was also considered mainly for comparison purposes, as it is widely used in industrial communication. The study found that all of these communication protocols are viable options, but OPC UA offers the most advantages in cases like the one in the project because it offers

real-time data exchange for machine-to-machine communication. The complete development of the connection with those communication servers falls out of the scope of the development of the thesis but it are necessary to understand the framework in which the project is produced. The results of this extensive analysis provided valuable insight into the most suitable communication protocols for future research.

A consistent data analysis was performed during the project to understand the initial performance of the implementation in site. The baseline value obtained for the KPIs studied in this research show that there is potential for automation in additive manufacturing and give further information in the areas to improve.

In conclusion, cloud workflow controllers have high potential to improve Additive Manufacturing efficiency and automation with Industrial Internet of Things solutions. This research provides valuable advancements and insights into automation in additive manufacturing technology and emphasizes the need for further research and development for the expansion of Industry 4.0.

4.1 Future work

The research presented in this Master's thesis represents a significant step forward in the investigation in automation in additive manufacturing. However, there is still significant room for further exploration and development in this field. Future work in this area could focus on several directions to further advance the research and implementation on cloud workflow controllers or for automation of additive manufacturing in general, including:

1. Further investigation of Behavior Trees as a model of computation. Even though BTs have shown promising results, there is still room for further investigation and enhancement of their application in this project or similars. In that sense, one possibility could be to explore the utilization of different libraries to create behavior trees, which could offer additional features or advantages over the current framework chosen.

2. Implementation of automatic data analytics. As explained in the section, 3.1.3.10 at present, the ability to automatically analyze the data from the production line has not yet been implemented. A possible future improvement for the project could be to provide real-time insights about important performance metrics like production and defect rates. It could be also possible to detect and diagnose potential issues before they become critical, allowing for proactive maintenance and repairs,

3. Expansion of additional KPIs for analyzing the performance of the workflow controller in a real-world manufacturing setting. This could involve identifying new

metrics for evaluating the efficiency, accuracy, and cost-effectiveness of the system, as well as developing tools and techniques for collecting and analyzing data from the production line.

4. Further research on communication protocols implemented in IIoT, specifically in the context of additive manufacturing. Improving the application of IIOT solutions in AM might need further analysis and comparison between the different available protocols, as well as testing with new protocols that may better suit the specific requirements of additive manufacturing production lines.

5. Improve the deliverables of the project. Particularly with regard to in-site implementation of the controller at the factory there is still room for improvement when connecting to the plant's net and when setting necessary preconditions. As part of the effort to address these challenges, it's anticipated that the graphical user interface will be further developed, although some functionality is already in place, there are still various features and capabilities that require development.

6. Further testing and validation of the workflow controller in the BMW plant in Munich. The cloud workflow controller developed needs to be further deployed and tested in the real manufacturing scenario for the validation of its performance and effectiveness and to identify any necessary improvements. This will provide further insights into the practical applications and limitations of the controller in an industrial setting.

7. Scalability and Sustainability: As the number of connected devices and systems in the scheme continues to grow, it is important to consider the scalability and sustainability of the cloud workflow controller keeping in mind the possibilities of automation in additive manufacturing. Research into approaches to address these challenges will be advantageous to ensure the long-term viability and success of this technology in industrial settings.

Overall, future work in this area has the potential to further advance the state of the art in IIoT solutions for additive manufacturing, leading to more efficient, reliable, and cost-effective production processes.

Bibliography

[1] R. Horstkotte, B. Bruning, M. Prümmer, K. Arntz, and T. Bergs, *Determination of the level of automation for additive manufacturing process chains*. Hannover: publish-Ing, 2021. DOI: https://doi.org/10.15488/11257.

[2] D. S. Thomas and S. W. Gilbert, *Costs and cost effectiveness of additive manufacturing*. National Institute of Standards and Technology, 2014.

[3] P. M. Bhatt, R. K. Malhan, A. V. Shembekar, Y. J. Yoon, and S. K. Gupta, "Expanding capabilities of additive manufacturing through use of robotics technologies: A survey," *Additive Manufacturing*, vol. 31, p. 100 933, 2020. DOI: https://doi.org/10.1016/j.addma.2019.100933.

[4] Manufacturing Business Technology, *Horizontal and vertical integration in industry 4.0*, Manufacturing Business Technology, 2019. [Online]. Available: https://www.mbtmag.com/business-intelligence/article/13251083/horizontal-and-vertical-integration-in-industry-40.

[5] J. Butt, "Exploring the interrelationship between additive manufacturing and industry 4.0," *Designs*, vol. 4, no. 2, p. 13, 2020. DOI: https://doi.org/10.3390/designs4020013.

[6] F. Šproch, V. Schindlerová, and I. Šajdlerová, "Using 3d printing technology in prototype production to control the dimensions of complexly shaped products," *MANUFACTURING TECHNOLOGY*, vol. 20, no. 3, pp. 385–393, 2020. DOI: 10.21062/mft.2020.061.

[7] L. Seiffert, J. Sczodrok, J. Ghofrani, and K. Wieczorek, "Remote technologies as common practice in industrial maintenance: What do experts say?" *Telecom*, vol. 3, no. 4, pp. 548–563, 2022. DOI: https://doi.org/10.3390/telecom3040031.

[8] T. Stock and G. Seliger, "Opportunities of sustainable manufacturing in industry 4.0," *Procedia CIRP*, vol. 40, pp. 536–541, 2016. DOI: https://doi.org/10.1016/j.procir.2016.01.129.

[9] D. Mies, W. Marsden, and S. Warde, "Overview of additive manufacturing informatics: "a digital thread."," *Integrating Materials and Manufacturing Innovation*, vol. 5, no. 1, pp. 114–142, 2016. DOI: https://doi.org/10.1186/s40192-016-0050-7.

[10] W. Gao, Y. Zhang, D. Ramanujan, *et al.*, "The status, challenges, and future of additive manufacturing in engineering," *Computer Aided Design*, vol. 69, pp. 65–89, 2019. [Online]. Available: https://www.academia.edu/38520195/The_status_challenges_and_future_of_additive_manufacturing_in_engineering.

[11] F. Almada-Lobo, "The industry 4.0 revolution and the future of manufacturing execution systems (mes)," *Journal of Innovation Management*, vol. 3, no. 4, pp. 16–21, 2016. DOI: https://doi.org/10.24840/2183-0606_003.004_0003.

[12] J. L. Harris, P. Sunley, E. Evenhuis, R. Martin, A. Pike, and R. Harris, "The covid-19 crisis and manufacturing: How should national and local industrial strategies respond?" *Local Economy*, vol. 35, no. 4, pp. 403–415, 2020. DOI: 10.1177/0269094220953528.

[13] M. Cugno, R. Castagnoli, G. Büchi, and M. Pini, "Industry 4.0 and production recovery in the covid era," *Technovation*, vol. 114, p. 102 443, 2022. DOI: 10.1016/j.technovation.2021.102443.

[14] M. Petch, *3d printing community responds to covid-19 and coronavirus resources*, 3D Printing Industry, 2020. [Online]. Available: https://3dprintingindustry.com/news/3d-printing-community-responds-to-covid-19-and-coronavirus-resources-169.

[15] 3YOURMIND, *3yourmind*, https://www.3yourmind.com/production-and-quality-management, [Accessed January 17, 2023], 2023.

[16] H. Ibrahim, A. Jahadakbar, A. Dehghan, N. Moghaddam, A. Amerinatanzi, and M. Elahinia, "In vitro corrosion assessment of additively manufactured porous niti structures for bone fixation applications," *Metals*, vol. 8, no. 3, p. 164, 2018. DOI: https://doi.org/10.3390/met8030164.

[17] M. Nematollahi, G. Toker, S. E. Saghaian, *et al.*, "Additive manufacturing of ni-rich nitihf20: Manufacturability, composition, density, and transformation behavior," *Shape Memory and Superelasticity*, vol. 5, no. 1, pp. 113–124, 2019. DOI: https://doi.org/10.1007/s40830-019-00214-9.

[18] P. Košťál, A. Mudriková, and D. Michal, "Group technology in the flexible manufacturing system," *MATEC Web of Conferences*, vol. 299, p. 02 001, 2019. DOI: https://doi.org/10.1051/matecconf/201929902001.

[19] D. Lehmhus, T. Wuest, S. Wellsandt, *et al.*, "Cloud-based automated design and additive manufacturing: A usage data-enabled paradigm shift," *Sensors*, vol. 15, no. 12, pp. 32 079–32 122, 2015. DOI: 10.3390/s151229905.

[20] I. BMBF, *Bundesministerium für bildung und forschung - bmbf*, Accessed: January 20, 2023, 2011. DOI: https://www.bmbf.de/bmbf/de/forschung/digitale-wirtschaft-und-gesellschaft/industrie-4-0/industrie-4-0.

[21] W. Zhang, X. Deng, and J. Cai, "Intelligent automatic 3d printing machine based on wireless network communication," *Wireless Communications and Mobile Computing*, vol. 2021, pp. 1–14, 2021. DOI: https://doi.org/10.1155/2021/1778.

[22] U. Delli and S. Chang, "Automated process monitoring in 3d printing using supervised machine learning," *Procedia Manufacturing*, vol. 26, pp. 865–870, 2018. DOI: https://doi.org/10.1016/j.promfg.2018.07.111.

[23] P. Kellett, *Additive manufacturing and automation will 3d printing displace automation technologies?* Association for Advancing Automation, 2012. [Online]. Available: https://www.automate.org/userAssets/riaUploads/file/Additive_Manufacturing_and_Automation.pdf.

[24] Y. Wang, Y. Lin, R. Y. Zhong, and X. Xu, "Iot-enabled cloud-based additive manufacturing platform to support rapid product development," *International Journal of Production Research*, vol. 57, no. 12, pp. 3975–3991, 2019. DOI: https://doi.org/10.1080/00207543.2018.1516905.

[25] A. Shah, "Emerging trends in robotic aided additive manufacturing," *Materials Today: Proceedings*, vol. 62, pp. 7231–7237, 2022. DOI: https://doi.org/10.1016/j.matpr.2022.03.680.

[26] J. P. N. Freens, I. J. B. F. Adan, A. Y. Pogromsky, and H. Ploegmakers, "Automating the production planning of a 3d printing factory," in *2015 Winter Simulation Conference (WSC)*, 2015, pp. 2136–2147. DOI: 10.1109/WSC.2015.7408424.

[27] S. Martello, D. Pisinger, and D. Vigo, "The three-dimensional bin packing problem," *Operations Research*, vol. 48, no. 2, pp. 256–267, 2000.

[28] R. Dromey, "Genetic software engineering – simplifying design using requirements integration," in *IEEE Working Conference on Complex and Dynamic Systems Architecture*, Brisbane, 2001.

[29] D. Isla, *Gdc 2005 proceeding: Handling complexity in the halo 2 ai*, Game Developer, 2005. [Online]. Available: https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai.

[30] D. Isla, "Building a better battle: Halo 3 ai objectives," in *Game Developers Conference*, 2008.

[31] R. Ghzouli, S. Dragule, T. Berger, E. B. Johnsen, and A. Wasowski, *Behavior trees and state machines in robotics applications*, 2022. arXiv: 2208.04211 [cs.RO].

[32] S. W. Jones, "Onboard evolution of human-understandable behaviour trees for robot swarms," University of Bristol, 2020. [Online]. Available: https://research-information.bris.ac.uk/en/studentTheses/onboard-evolution-of-human-understandable-behaviour-trees-for-rob.

[33] M. Colledanchise, *Behavior trees in robotics*, Retrieved January 9, 2023, from https://www.diva-portal.org/smash/get/diva2:1078940/FULLTEXT01.pdf, 2017.

[34] M. Colledanchise and P. Ögren, "Behavior trees in robotics and ai: An introduction," *arXiv preprint arXiv:1709.00084*, 2017.

[35] D. Faconti, *Behaviortree.cpp library documentation*, https://www.behaviortree.dev, 2022.

[36] D. Faconti and Groot, *Groot*, https://github.com/BehaviorTree/Groot, 2019.

[37] R. A. Agis, S. Gottifredi, and A. J. García, "An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games," 2020.

[38] S. Cooper and S. Lemaignan, "Towards using behaviour trees for long-term social robot behaviour," in *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, IEEE, 2022, pp. 737–741.

[39] S. W. Jones, "Onboard evolution of human-understandable behaviour trees for robot swarms," Ph.D. dissertation, University of Bristol, 2020. [Online]. Available: https://research-information.bris.ac.uk/en/studentTheses/onboard-evolution-of-human-understandable-behaviour-trees-for-rob.

[40] A. Klockner, *Behavior trees for uav mission management*, DLR Technical Report, 2013. [Online]. Available: https://elib.dlr.de/91679/1/kloeckner2013behavior.pdf.

[41] E. W. Dijkstra, "Letters to the editor: Go to statement considered harmful," *Communications of the ACM*, vol. 11, no. 3, pp. 147–148, 1968. DOI: 10.1145/362929.362947.

[42] T. G. Tadewos, L. Shamgah, and A. Karimoddini, "Automatic safe behaviour tree synthesis for autonomous agents," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, IEEE, 2019, pp. 2776–2781.

[43] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ogren, "Towards a unified behavior trees framework for robot control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 5420–5427.

[44] A. J. Champandard and P. Dunstan, "The behavior tree starter kit," in *Game AI Pro 360*. CRC Press, 2019, pp. 27–46.

[45] G Legarda Herranz, "Evolution of behaviour trees for collective transport with robot swarms," Universitat Politècnica de Catalunya, 2021.

[46] S. Jones, M. Studley, S. Hauert, and A. Winfield, "Evolving behaviour trees for swarm robotics," in *Distributed Autonomous Robotic Systems*, Springer, 2018, pp. 487–501.

[47] C. Rodríguez, M. Baez, F. Daniel, *et al.*, "Rest apis: A large-scale analysis of compliance with principles and best practices," in *Lecture Notes in Computer Science*, Springer, 2016, pp. 21–39.

[48] OPC UA Foundation, *Home page*, https://opcfoundation.org/, Accessed: March 6, 2023, 2013.

[49] M. Ladegourdie and J. Kua, "Performance analysis of opc ua for industrial interoperability towards industry 4.0," *IoT*, vol. 3, no. 4, pp. 507–525, 2022. DOI: 10.3390/iot3040027.

[50] H. Bauer, S. Höppner, C. Iatrou, *et al.*, "Hardware implementation of an opc ua server for industrial field devices," *ArXiv [Cs.AR]*, 2021. arXiv: 2105.00789 [cs.AR].

[51] P. Drahos, E. Kucera, O. Haffner, and I. Klimo, "Trends in industrial communication and opc ua," in *2018 Cybernetics & Informatics (K&I)*, IEEE, 2018, pp. 1–6.

[52] T. Friihwirth, F. Pauker, A. Fernbach, I. Ayatollah, W. Kastner, and B. Kittl, "Guarded state machines in opc ua," in *IECON 2015-41st Annual Conference of the IEEE Industrial Electronics Society*, IEEE, 2015, pp. 004 187–004 192.

[53] H. Derhamy, J. Ronnholm, J. Delsing, J. Eliasson, and J. van Deventer, "Protocol interoperability of opc ua in service oriented architectures," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, IEEE, 2017, pp. 44–50.

[54] *Step 7 professional (tia portal)*, https://mall.industry.siemens.com/mall/es/es/Catalog/Products/10314843, Accessed: January 20, 2023, 2023.

[55] *Umati - universal machine technology interface - connecting the world of machinery*, https://umati.org/, Accessed: Jan 5, 2023.

[56] R. F. Maia, Á. J. Bálsamo, G. A. W. Lopes, A. A. Massote, and F. Lima, "Evaluation of opc-ua communication in an autonomous advanced manufacturing cell implementation," *Gestão & Produção*, vol. 27, no. 4, 2020. DOI: 10.1590/0104-530x5414-20.

[57] *MQTT - the standard for IoT messaging*, https://mqtt.org/, Accessed: March 6, 2023.

[58] C. Bayılmış, M. A. Ebleme, U. Çavuşoğlu, K. K"uç"uk, and A. Sevin, "A survey on communication protocols and performance evaluations for internet of things," *Digital Communications and Networks*, vol. 8, no. 6, pp. 1094–1104, 2022. DOI: 10.1016/j.dcan.2022.03.013.

[59] P. Gupta and I. O. Prabha, "A survey of application layer protocols for internet of things," in *2021 International Conference on Communication Information and Computing Technology (ICCICT)*, IEEE, 2021, pp. 1–6. DOI: 10.1109/ICCICT51719.2021.9352535.

[60] F. Azzedin and T. Alhazmi, "Secure data distribution architecture in iot using mqtt," *Applied Sciences (Basel, Switzerland)*, vol. 13, no. 4, p. 2515, 2023. DOI: 10.3390/app13042515.

[61] A. Fisher, G. Srivastava, and R. Bryce, "Mqttg: An android implementation," in *arXiv [cs.NI]*, 2019.

[62] R. Giambona, A. E. C. Redondi, and M. Cesana, "Mqtt+: Enhanced syntax and broker functionalities for data filtering, processing and aggregation," in *Proceedings of the 14th ACM International Symposium on QoS and Security for Wireless and Mobile Networks*, ACM, 2018.

[63] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–29, 2019.

[64] Eiger.Io, *Eiger api v3*, https://www.eiger.io/developer, Accessed on: February 4, 2023, n.d.

[65] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[66] R. Schiekofer, A. Scholz, and M. Weyrich, "Rest based opc ua for the iiot," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, vol. 1, 2018, pp. 274–281.

[67] M. Abdellatif, R. Tighilt, A. Belkhir, N. Moha, Y.-G. Guéhéneuc, and É. Beaudry, "A multi-dimensional study on the state of the practice of rest apis usage in android apps," *Automated Software Engineering*, vol. 27, no. 3–4, pp. 187–228, 2020. DOI: 10.1007/s10515-020-00272-9. [Online]. Available: https://doi.org/10.1007/s10515-020-00272-9.

[68] B. G. Wolde and A. S. Boltana, "Rest api composition for effective testing the cloud," *Journal of Applied Research and Technology*, vol. 19, no. 6, pp. 676–693, 2021.

[69] N. Petrovic, M. Radenković, S. Cvetkovic, and D. Rancic, "Model-driven automated gmock test generation for automotive software industry," *arXiv preprint arXiv:2110.09359*, 2021.

[70] S Kumar, "Supercharge your rest apis with protobuf," *The Startup*, 2019. [Online]. Available: https://medium.com/swlh/supercharge-your-rest-apis-with-protobuf-b38d3d7a28d3.

[71] C. Hottel, *Efficient iiot communications: A comparison of MQTT, OPC-UA, HTTP, and modbus*, https://cirrus-link.com/efficient-iiot-communications-a-comparison-of-mqtt-opc-ua-http-and-modbus/, [Online; accessed February 4, 2023], 2019.

[72] J. Wytrebowicz, K. Cabaj, and J. Krawiec, "Messaging protocols for iot systems-a pragmatic comparison," *Sensors (Basel, Switzerland)*, vol. 21, no. 20, p. 6904, 2021. DOI: https://doi.org/10.3390/s21206904.

[73] F. M. for Economic Affairs and Energy, *Rami 4.0: Reference architecture model for industrie 4.0*, https://www.bmwi.de/Redaktion/EN/Publikationen/Digitale-Welt/reference-architecture-model-industrie-4-0.html, 2015.

[74] MagneticValley, *Mqtt: The solution for our real time data transfer needs*, https://www.intermagnet.org/publications/dinant2016/mqtt.pdf, Accessed on February 6, 2023, 2016.

[75] J. Bartnitsky, *Http vs mqtt performance tests*, https://flespi.com/blog/http-vs-mqtt-performance-tests, Accessed on February 7, 2023, 2018.

[76] T. Kušević, D. Blažević, and T. Keser, "Comparison functionalities of http and mqtt protocols," in *31st International Conference on Organization and Technology of Maintenance (OTO 2022)*, Springer International Publishing, 2023, pp. 45–56.

[77] I. Craggs, *MQTT vs HTTP for IoT*, https://www.hivemq.com/blog/mqtt-vs-http-for-iot/, [Accessed on: February 7, 2023], 2022.

[78] C. Lesjak, D. Hein, M. Hofmann, *et al.*, "Securing smart maintenance services: Hardware-security and tls for mqtt," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, IEEE, 2015, pp. 1243–1250. DOI: 10.1109/INDIN.2015.7281904.

[79] *OPC UA + MQTT = A popular combination for IoT expansion*, https://opcconnect.opcfoundation.org/2019/09/opc-ua-mqtt-a-popular-combination-for-iot-expansion/, Accessed on February 6, 2023, 2019.

[80] C. L. Solutions, *IIoT Protocols: Comparing OPC UA to MQTT*, https://cirrus-link.com/wp-content/uploads/2020/07/White-Paper_IIoT-Protocols-Comparing-OPC-UA-to-MQTT.pdf, Accessed: March 6, 2023, n.d.

[81] T. Atanasijevic, *HOW TO CREATE A SUCCESSFUL USER EXPERIENCE (UX) ON THE HAPPY PATH AND THE UNHAPPY PATH*, 2020. DOI: 10.13140/RG.2.2.10387.71207.