



UNIVERSITÀ
DI TRENTO
Dipartimento di
Ingegneria Industriale



Digital
MASTER SCHOOL

UNIVERSITÉ
CÔTE D'AZUR



Master's Degree in Mechatronics Engineering

FINAL DISSERTATION

CLOUD WORKFLOW CONTROLLER FOR IIOT SOLUTIONS IN ADDITIVE MANUFACTURING

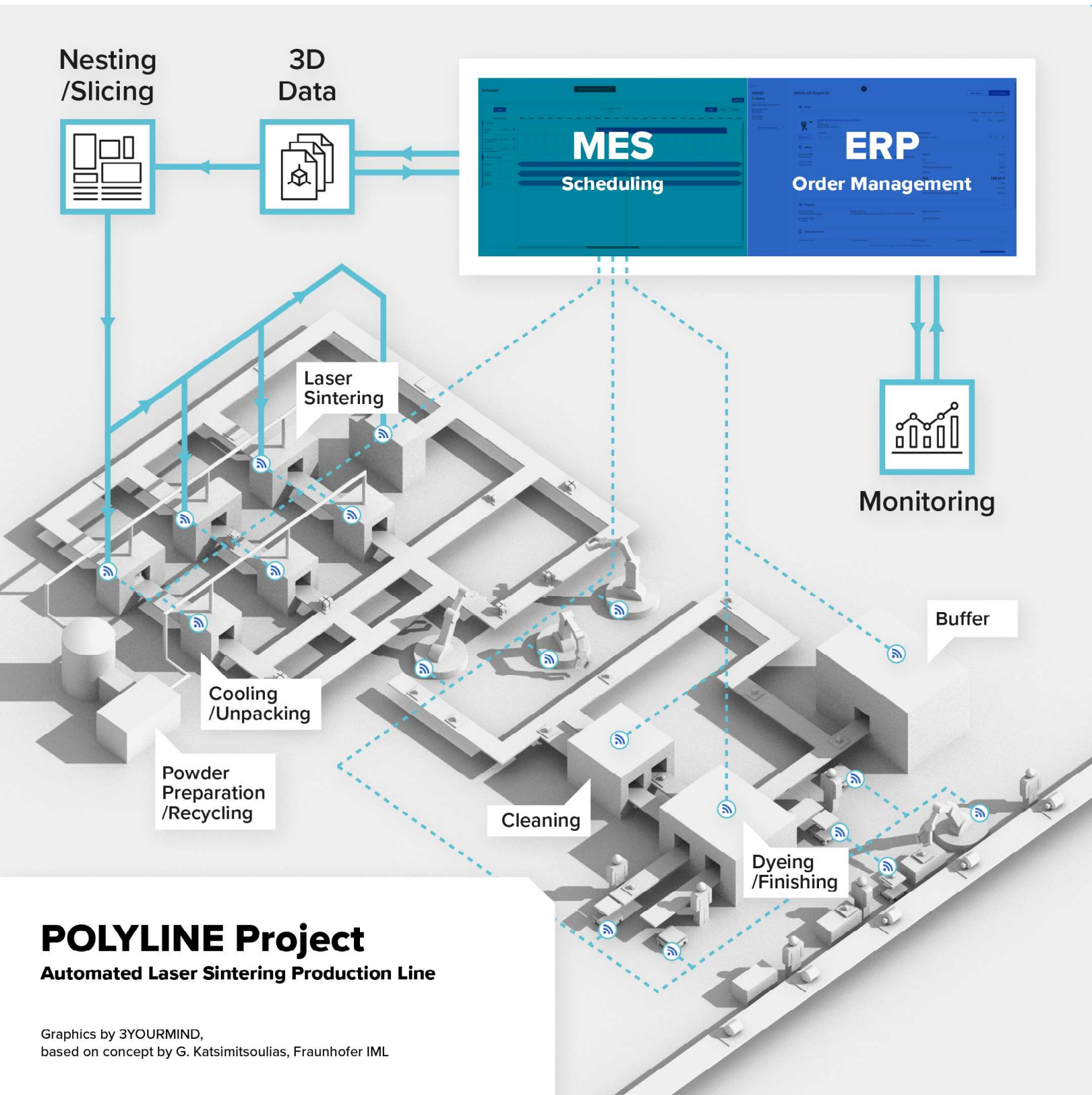
In the context of the POLYLINE project

Supervisor UniTrento
Daniele Fontanelli

Student
Zaida Brito Triana, [MAT. 230400]

Supervisor Company
Simone Dal Poz

ACADEMIC YEAR 2021/2022





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 in additive manufacturing plant of BMW in Munich.

This research is centered around exploring potential models of computation that can be leveraged to construct a logic-based model of the production line designed in the project, in order to do so, the research presented in this thesis investigates some of the most commonly used models for these purposes and examines the advantages that Behavior Trees (BTs) have offered for similar projects to this one in the fields of robotics, manufacturing, and gaming. Specifically, the research delves into the ways in which BTs demonstrate potential for effectively organizing and managing the complex processes and decision-making situations involved in the operation of a production line. Via a comprehensive examination of the capabilities and limitations of BTs and other relevant models, this thesis aims to provide insight into the most suitable approach for constructing a logic-based model of the production line in question.

Furthermore, this research underwent a comprehensive study of the principal communication protocols implemented 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 a comprehensive understanding of the underlying automation functionality intrinsic to the project. While the primary protocol employed in the machines involved 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 research.

Additionally, a thorough data analysis was conducted to gain a comprehensive understanding of the initial performance of the implementation site. This analysis aimed to identify key performance indicators (KPIs) and determine their initial values. The analytics were found to be an invaluable resource for comprehending the overall performance of the system, thereby facilitating informed decision-making regarding the most suitable approach for future development.

The workflow controller has been tested in a mock environment prior to its deployment in the BMW plant. 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.

This Master's thesis presents a comprehensive overview of the current state of research in the field of industrial automation, including a novel cloud workflow controller design and its capabilities. The research contained within this thesis represents a significant contribution to the field, providing valuable insights and advancements in the understanding and application of IIoT in additive manufacturing.



Contents

Abstract.....	1
List of figures.....	5
1 Introduction.....	8
1.1 <i>What is the polyline project?.....</i>	<i>10</i>
1.2 <i>Goals of this thesis/SOLUTIONS.....</i>	<i>10</i>
2 Literature review/theory.....	13
2.1 <i>Automation in Additive Manufacturing.....</i>	<i>13</i>
2.2 <i>Behavior Trees.....</i>	<i>16</i>
2.2.1 Behavior Tree theory.....	17
2.2.1.1 Common node types.....	18
2.2.1.2 Blackboard.....	21
2.2.1.3 Challenges with Behavior Trees.....	22
2.2.2 Behavior Trees vs Finite State Machines.....	22
2.2.2.1 Hierarchical Finite State Machines.....	24
2.2.3 State of the art in Behavior Trees.....	25
2.3 <i>Communication Protocols in IIOT.....</i>	<i>27</i>
2.3.1 OPC-UA.....	27
2.3.1.1 UMATI.....	31
2.3.2 MQTT.....	33
2.3.3 Http.....	35
2.3.3.1 REST API.....	35
3 Methodology and results.....	38
3.1 <i>Project outline.....</i>	<i>39</i>
3.2 <i>Trigger utility - Markforged reports.....</i>	<i>40</i>
3.3 <i>Comparison between protocols.....</i>	<i>43</i>
3.4 <i>Behavior Tree implementation.....</i>	<i>45</i>
3.4.1 Workflow design.....	47
3.4.2 Machines and Behavior Tree Nodes.....	55
3.4.3 Communication nodes.....	58
3.4.4 Control nodes.....	61
3.5 <i>Mock environment results.....</i>	<i>63</i>
3.6 <i>Analytics - On site results.....</i>	<i>65</i>
4 Future work.....	67



5 Conclusions.....	69
6 Bibliography.....	71



List of figures

Figure 1. Industrial revolution timeline, Source (Rao et al., 2020).....	9
Figure 2. Schematic representation of an example line production of the laser-sintering process, highlighting the Manufacturing Execution System and the Enterprise Resource Planning, center focus on the thesis.....	11
Figure 3. Pillars of Industry 4.0, retrieved from (Butt, 2020).....	15
Figure 4. OPC UA Server Interface. (Source: https://www.kb-controls.io/post/plc-plc-communication-with-opc-ua).....	28
Figure 5. Configuration required to make a machine tool compatible with UMATI (Source: Edition based on the diagram specifications published in VDW).....	32
Figure 6, MQTT schema. Source: https://www.sciencedirect.com/science/article/pii/S2352864822000347	34
Figure 7. Schema of the connections and functionalities in the project.....	40
Figure 8. Communication architecture of the implementation with the Markforged printer.....	41
Figure 9. JSON vs Protobuf performance comparison.....	42
Figure 10. Schema of the Markforged communication.....	43
Figure 11. Log replay from the testing of one subtree.....	45
Figure 12. A study of the trend of usage of state machine and behavior tree languages in open-source projects on GitHub over time. Extracted from (Ghzouli et al., 2022).....	47
Figure 13. Workflow schema.....	48
Figure 14. Nabertherm OpenDoor behavior tree.....	48
Figure 15. Main behavior tree of the project.....	50
Figure 16. Transfer station schema.....	52
Figure 17. Transfer station schema.....	52
Figure 18. Schema of the workflow.....	54
Figure 19. Subtree representation day 2 of output nodes.....	60
Figure 20. Subtree representation, example of input nodes.....	60





Keywords

Additive Manufacturing (AM)
Automated Guided Vehicles (AGVs)
Behavior Tree (BT)
Cloud Manufacturing (CM)
Finite State Machine (FSM)
Flexible manufacturing systems (FSM)
Graphical User Interface (GUI)
Enterprise Resource Planning (ERP)
Hierarchical Finite State Machine (HFSM)
Industrial Internet of Things (IIOT)
Industrial Message Queuing Telemetry Transport (MQTT)
Key Performance Indicators (KPIs)
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)
Protocol Buffers (Protobufs)
Representational State Transfer (REST)
Universal Machine Type Interface (UMATI)



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 (Horstkotte et al., 2021). 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 (Thomas & Gilbert, 2014), as well as allowing for greater adaptability for OEMs and manufacturers (Bhatt et al., 2020).

Additive manufacturing integration into conventional lines can only be achieved to a limited extent at present, both vertically and horizontally (“Horizontal and vertical integration in industry 4.0,” 2019) (Butt, 2020). Initially, this is attributed to the specific AM production steps (e.g., curing, post-processing, slicing, batch production time), but a closer examination reveals it's related to the overall low implementation of automation techniques in AM manufacturing and post-production processes. (Butt, 2020). Due to this, production intervals are elongated, and there is a great need for manual work involved not only to mass-produce but to prototype as well. The discontinuity of the digital data chain along the horizontal process chain results in a lack of comprehensive monitoring along the processes, hindering integration 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 (Horstkotte et al., 2021) in the same remote technologies to enable automatization are still yet to be implemented in many industries remains restricted in a general sense, as evidenced by the findings of the study conducted by Seiffert et al. (2022).

Until now, the research and implementation of digital solutions for managing production lines in Additive Manufacturing have been quite limited (Mies et al., 2016). The integration and collaboration with other technologies needed, such as autonomous robots, cloud control, and integration, are shown to be key for the optimal implementation of Industry 4.0; the path to smart and flexible manufacturing goes through the development and integration with automation (Butt, 2020). The ongoing process of globalization faces the challenge of meeting the increasing worldwide demand for capital and consumer goods while ensuring sustainable development across social, environmental, and economic dimensions. To address this challenge, industrial value creation must prioritize sustainability. The

current landscape presents numerous opportunities for Additive Manufacturing, as studied in (Stock & Seliger, 2016).

The rise of Industry 4.0 and the increasing popularity of emerging technologies has created a unique space for the transformation of traditional manufacturing methods. (Gao et al., 2019). 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. This paper 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 sustainable manufacturing in Industry 4.0, as seen in Figure 1. The prediction of Industry 4.0 has opened up opportunities for developing roadmaps for manufacturing operations, especially for manufacturing IT systems. 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 (Almada-Lobo, 2016).

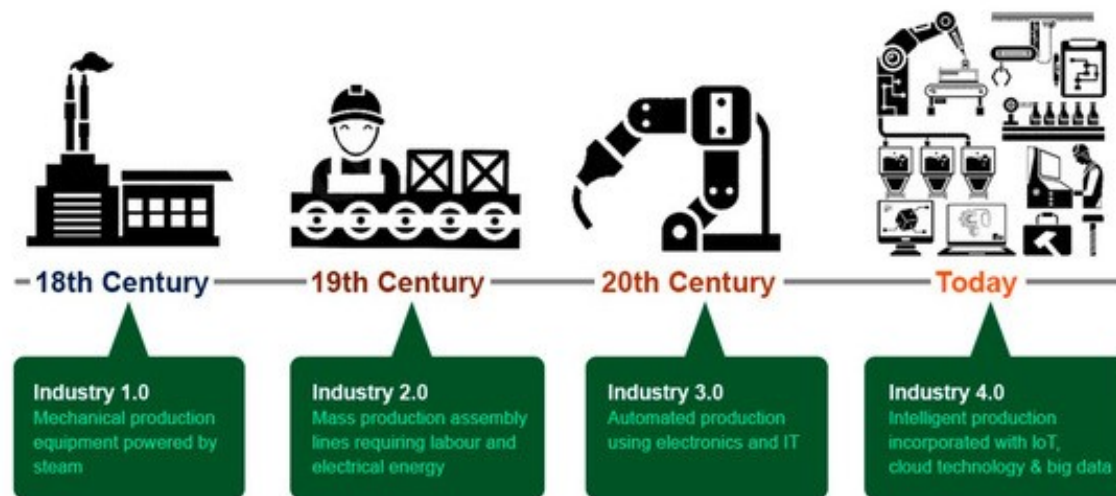


Figure 1. Industrial revolution timeline, Source (Rao et al., 2020)

It is worth highlighting that the COVID-19 pandemic has emphasized the criticality of rapidly and efficiently responding to real-time changes in manufacturing operations, as the traditional manufacturing methods might be getting behind in their adaptability horizons (Petch, 2020). 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

automatization, data management, and the Internet of Things, will help to improve manufacturing efficiency and reduce lead time.

1.1 What is 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 and low degrees of automation for physical handling and transport processes and efficient integration the entire additive production process into existing mass production. The partners involved in this project include BMW, which is the location of all the machines; EOS, Grenzebach, Krumm-Tec, DyeMansion, and Nabetherm, which are the manufacturers of the machines; Additive Marking, which provides a software solution for applying serialization to the 3D geometry of the parts; the University of Augsburg, which designed the workflow for the project; and 3YOURMIND, that had to wait for the manufacturers to develop the necessary machines the necessary documentation from the partners to then correctly implement the entire workflow.

1.2 Goals of this thesis/SOLUTIONS

The overarching objective of this thesis is to give an overview of the state of the art in Industry 4.0, with a specific focus on the domain of industrial automation and additive manufacturing. The goal is to conduct a comprehensive analysis of the current advancements and developments in this field, in order to provide a clear understanding of the state of the art. The research aims to provide valuable insights and information that can be used to guide future development and innovation in the field of industrial automation and additive manufacturing.

The main goal of this research work is to collaborate in the creation of a cloud workflow controller within the POLYLINE project, which aims at the

© G. Katsimitsoulis, Fraunhofer IML (edited)

pg. 11



projects similar to this one in the fields of robotics, manufacturing, and gaming. Through a thorough analysis of the capabilities and limitations of BTs and other relevant models, this thesis aims to provide insight into the most suitable approach for constructing a logic-based model of the production line in question.

In the second segment of this dissertation, an in-depth examination will be carried out to assess the communication protocols utilized in the Industrial Internet of Things (IIOT) in order to scrutinize the proposed connections for completing the workflow. The principal protocol implemented in the research will be 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 furnish invaluable insight into the most appropriate communication protocols for future research.

In order to evaluate the performance of the workflow and determine the system's capacity to sustain the workload generated by the production process, a mock environment will be created for this purpose. The workflow controller will undergo testing in this simulated environment prior to its deployment in the BMW plant. The results of these tests will demonstrate that the workflow controller possesses the capability to support the workload generated by the production process, thereby ensuring a seamless transition between the various stages of the production line.

Furthermore, a comprehensive data analysis was performed with the goal of conducting analysis with the available data to gain a deep understanding of the initial performance of the implementation site. The objective of this analysis was to recognize key performance indicators (KPIs) and establish their initial values. These analytics proved to be a useful tool in comprehending the overall performance of the system, thereby allowing informed decisions to be made regarding the best approach for future development.

2 Literature review/theory

This section aims to present a comprehensive examination of the current state-of-the-art in the automation of additive manufacturing. The section will begin by highlighting the current challenges and possibilities in the field, followed by an in-depth analysis of the theoretical principles and concepts of behavior trees and a comparison with other commonly used methodologies in the realm of manufacturing automation. Furthermore, the current state-of-the-art of behavior trees in similar applications will also be examined. Finally, the section will conclude with a discussion of the most prevalent communication protocols currently utilized within the Industrial Internet of Things (IIoT) and their current state of advancement. In sum, this section aims to provide a thorough understanding of the current state-of-the-art in the field of additive manufacturing automation and related areas, with a specific focus on the utilization of behavior trees and communication protocols within the IIoT. This serves as the theoretical foundation upon which this research is based.

2.1 Automation in Additive Manufacturing

Additive Manufacturing is a process of building 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. AM processes refer to techniques that build up a surface by adding material, resulting in varying part quality, density, and geometric accuracy (Ibrahim et al., 2018), (Nematollahi et al., 2019). 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 steps in automation mainly under Flexible manufacturing systems (FSM)(Košťál et al., 2019). AM provides greater design freedom, improved efficiency, and increased sustainability compared to traditional methods.

It is crucial to analyze the advancements in automation within additive manufacturing 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. First introduced in 2011 with the goal of promoting economic growth in Germany (Industrie 4.0 - BMBF, 2011), 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. There is a significant correlation between the concept of Industry 4.0 developed in Germany and the Industrial Internet 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 that the implementation of wireless communication network intelligent automatic 3D printing machinery increases printing efficiency by 15% and saves economic cost by 20%, highlighting the practicality of the research findings (Zhang et al., 2021). Additionally, there have been noteworthy advancements in the field of additive manufacturing, such as the incorporation of machine learning, as demonstrated in (Delli & Chang, 2018).

The concept of Industry 4.0, also known as the fourth industrial revolution, is a technology-driven shift in manufacturing that aims to improve various aspects of human life. It involves the integration of smart automation, decision-making, knowledge, problem-solving, self-diagnosis, self-configuration, and self-automation in industries (Shah, 2022). This technology has the potential to influence various levels of the manufacturing process, the end-users, designers, managers, and all employees involved in the manufacturing process and supply chains. This study presents the application of a decision tree algorithm for the purpose of monitoring energy consumption in machines and appliances, as well as predicting future behavior and detecting anomalous activity. The effectiveness of the system is assessed and compared with other established methodologies, revealing a noteworthy efficiency rate of 78%. However, the implementation of Industry 4.0 also faces standardization issues, security issues, resource planning challenges, legal issues, and changes in business paradigms. The efficacy of Industry 4.0 relies on the performance of the complete production chain, involving all stakeholders ranging from manufacturers to end-users.

As seen in this article (Kellett, 2012) 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", and that new market opportunities will open up 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. (Wang et al., 2019). Recently, there has been an increase in collaboration between robotics and additive manufacturing for the purpose of enhancing production (Bhatt et al.,

2020). Additionally, additive manufacturing has opened up promising sustainable prospects, as demonstrated by (Stock & Seliger, 2016).

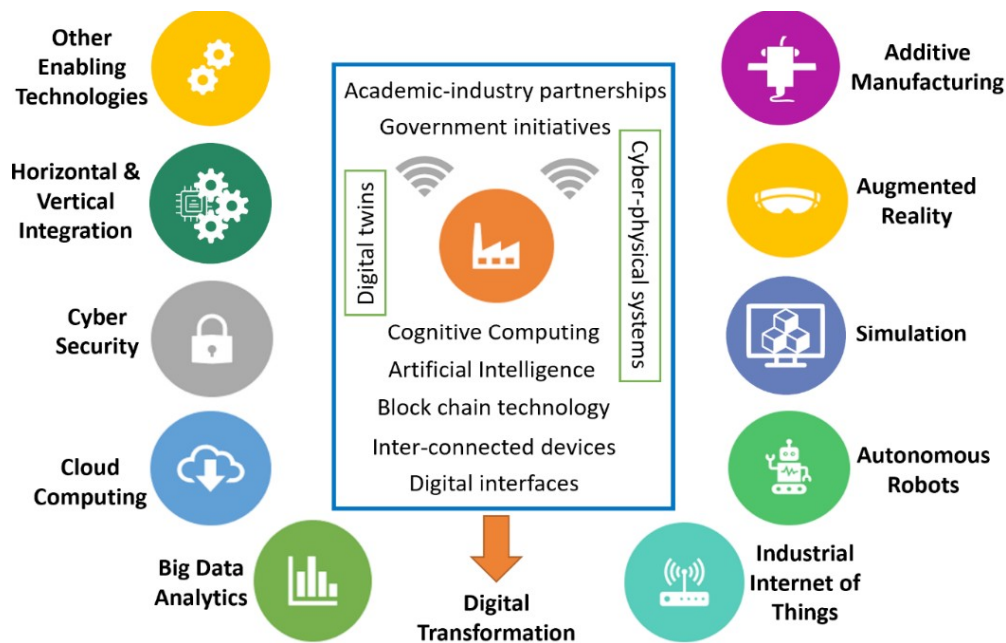


Figure 3. Pillars of Industry 4.0, retrieved from (Butt, 2020)

There is limited research available on the automation of additive manufacturing facilities, however, this study from 2015 (Freens et al., 2015) is highly relevant to the focus of the present investigation. 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. The production planning process involves composing printer trays by allocating objects to batches and determining their positions in the tray using a 3D packing algorithm. The study proposed an extension of the classical one-dimensional bin packing problem that takes into account specific requirements of high-volume 3D printing environments. The method was tested in a simulation study using one month of production data from Shapeways, which showed an increase in printer capacity and shorter waiting times. The main drawback of the method is difficulty in accurately estimating the print time of batches. Future work should focus on designing a more elaborate simulation study to optimize object size mixtures for more accurate print time estimation.

2.2 Behavior Trees

A Behavior Tree is a hierarchical structure that organizes the sequencing of tasks for an autonomous agent, such as a robot or a virtual entity in a computer game. It is similar to a finite state machine, but it allows for more complex decision-making and task switching by breaking down tasks into smaller sub-tasks that can be organized into a tree-like structure.

Behavior Trees (BTs) are a hierarchical decision-making system that provides a structured approach to designing and implementing autonomous systems in computer programming and robotics. The system was created with the intention of providing a method for describing the behavior of non-player characters (NPCs) in computer games. These NPCs 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, with the publication of his paper defining the key concepts and applications of the system. However, one of the first significant implementations of BTs was in the development of the Halo 2 and Halo 3 video games (Isla, 2005, Isla, 2008). This demonstrated the potential of BTs in creating intelligent, autonomous systems that can react to changing environments.

Recently, the use of BTs in the robotics and control community has gained significant attention, as seen in the publication of research papers and books dedicated to the topic. Two Ph.D. theses, "Onboard Evolution of Human-Understandable Behaviour Trees for Robot Swarms" by Jones.SW and "Behavior Trees in Robotics" by Colledanchise. M, will be used as the basis for discussion and analysis in this research paper. The latter thesis served as a precursor to the highly regarded book "Behavior Trees in Robotics and AI," which will also be referenced in this research.

The example presented in this research paper was implemented using the "BehaviorTree. CPP" framework (Faconti, 2022), written in C++ 17, and the "Groot" editor (Faconti, 2019). Both of these programs are maintained and regularly updated by Faconti.D and have been instrumental in the implementation of BTs in this field. They provide a powerful toolset for creating and visualizing behavior trees, making the process of designing and implementing autonomous systems more efficient and user-friendly.

2.2.1 Behavior Tree theory

Behavior trees are a type of decision tree used in artificial intelligence and robotics to specify the behavior of a system. They are composed of a hierarchy of nodes, each of which represents a specific behavior or action. The tree is traversed from the top down, and the actions of the nodes are executed in order based on the results of previous actions and the conditions specified in the tree.

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 might specify that a robot should pick a printed component if there is one available, but if the robot encounters that the piece is stuck, it should try to go around the obstacle or seek help from a human operator.

Behavior trees are a useful tool for specifying complex behaviors because they allow for a clear and concise representation of the decision-making process and enable easy modification of the behavior by modifying the tree structure. They are also easy to understand and debug, making them a feasible choice for specifying the behavior of robots and other intelligent systems

Corresponding to the matter in this research, we can 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 object, stop, or perform a gesture. A behavior tree (BT) begins by running its root node, which sends out signals at a specific frequency called ticks. These ticks are then passed on to the root node's children. A node can only be executed if it receives these ticks. When the root of a tree sends a tick 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  $i \leftarrow 1$  to  $N_{child}$  do
2:    $childReturn \leftarrow Tick(C_i)$ 
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,

Algorithm 2 Selector or Fallback node

```
1: for  $i \leftarrow 1$  to  $N_{child}$  do
2:    $childReturn \leftarrow Tick(C_i)$ 
3:   if  $childReturn = RUNNING$  then
4:     return  $RUNNING$ 
5:   if  $childReturn = SUCCESS$  then
6:     return  $SUCCESS$ 
7:   end if
8: end if
9: end for
10: return  $FAILURE$ 
```

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 elucidates the underlying logic of the node.

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 *RetryUntilSuccessful*.

Outline of the general characteristics of control execution nodes:

Algorithm 3 Decorator node

RetryUntilSuccessful with N attempts

```
1: for  $i \leftarrow 1$  to  $N_{child}$  do
2:   for  $n \leq N$  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  $FAILURE$ 
```

Action nodes: These nodes are the most common type of leaf node, and they represent a specific action or behavior that should be performed. When it receives a tick, an action node will execute a command. 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 (as described 3.4 Section) and modify the values of certain variables by writing to them.

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. They may read from the blackboard, but they cannot modify its contents.

The table above, inspired by (Colledanchise & Ögren, 2017), 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 $\geq N$ 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 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 a set of variables known as the blackboard. The blackboard mechanism facilitates the integration of events by allocating memory for specific tasks or by translating events into lower-level mission plan execution status. These statuses can be stored in the blackboard and reset after processing appropriate behavior tree tasks. (Agis et al., 2020)

In the context of behavior trees, a blackboard is a shared memory space that is used to store information that is relevant to the behavior tree and its nodes. The blackboard is accessible to all nodes in the behavior tree and allows them to share and access information in a centralized manner. This can be used to facilitate communication between different nodes, and to store information such as the current state of the system, the status of various tasks, or the results of sensor data.

The blackboard typically contains several different data structures, such as variables, flags, and data tables, that are used to store information in a structured manner. The blackboard can also include methods for reading, writing, and modifying the data stored within it. This allows nodes to access and manipulate the data in a consistent and predictable way.

Blackboards are useful in behavior trees because they allow the nodes to share information and coordinate their actions without the need for direct communication. This makes it possible to decouple the logic of different nodes and to create a more modular and reusable behavior tree. It also allows the behavior tree to adapt to changing circumstances by modifying the data stored in the blackboard, rather than by modifying the structure of the behavior tree itself.

In the shapes project (Cooper & Lemaignan, 2022), we 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 (Jones, 2020) 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

Implementing Behavior Trees can present certain challenges, particularly with single-threaded programming. However, once built, the engines can be reused with software libraries. It is also important to mention that BTs are designed around conditions, not states, and the switching occurs between tasks on each tick rather than events. Despite the availability of BT development tools, they are not as advanced as those for finite state machines (FSMs) and in the same line the software for BTs is less sophisticated compared to FSM software. Some of the limitations of BTs include difficulties in evaluating large trees and a lack of decision-making models. However, newer techniques like Utility AIs hold potential for creating advanced artificial intelligence in future applications.

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 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 (FSMs)." (Klockner, 2013)

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 shown in the demo developed in this paper, when a robot 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 (Colledanchise & Ögren, 2017), 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 (Dijkstra, 1968), that was utilized in early programming languages. The Goto statement serves as an exemplar 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 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 (Cooper & Lemaignan, 2022) (Tadewos et al., 2019), and more insights on the topic will be examined in greater detail within 2.2.3 Section.

2.2.2.1 Hierarchical Finite State Machines

Hierarchical Finite State Machines (HFSMs), were created to address some limitations of traditional FSMs. In an HFSM, a state can contain one or more sub-states and 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. These transitions can reduce the number of transitions needed by connecting two super-states instead of connecting a larger number of sub-states individually. Each super-state has a designated starting sub-state that is activated when transitioning to that super-state.

One potential disadvantage of using Hierarchical Finite State Machines (HFSMs) over Behavior Trees is complexity. As the number of states and transitions in an HFSM grows, the system can become increasingly complex and difficult to understand. This can make it harder to maintain and update an autonomous system, especially as the system becomes more intricate.

Additionally, the rigid structure of HFSMs can limit their ability to handle more complex or dynamic behavior, like in a robot swarm. Behavior Trees are designed to be more flexible in handling complex and dynamic behavior. They use a hierarchical tree structure, which allows for more intuitive visualization of the flow and logic of the system. The tree structure 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 AI systems depending on the requirements of the system. However, for more complex or dynamic systems, like the one presented in this research, BTs are often considered to be 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" (Marzinotto et al., 2014). Since then, attempts were made to standardize the basis of behavior trees, as seen in (Champanand & Dunstan, 2019). 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.

In this research conducted in 2021 by Legarda.H 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 first research (Jones, 2020), linked as well with another earlier publication (S. Jones et al., 2018), puts the focus on the issues of designing optimal and readable systems. 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; firstly, adaptability represents a crucial aspect to be considered, for the system described to possess 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 present era of society finds itself on the brink of a new epoch, as the processing power and connectivity of industrial embedded devices continues to increase, paving the way for a multitude of innovative applications that have the potential to alter the way manufacturing and consumption are approached. 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 a plethora of protocols developed in various domains, as stated in (Rodríguez et al., 2016). 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 REST API (along with HTTPS).

The objective of this section is to provide a comprehensive examination of the underlying principles behind the most widely employed communication protocols, which will be subsequently subjected to comparative analysis in the experimental section of this report. The primary aim is to provide the theoretical foundation necessary to comprehend the conceptual framework within which this study is situated, thereby endowing it with a comprehensive significance.

2.3.1 OPC-UA

Given that OPC UA is the primary communication protocol employed in this project and the only one that has been evaluated on-site, it is advisable to initiate the communication protocols analysis by focusing on it. By establishing the fundamental theoretical concepts, it becomes possible to examine in subsequent sections the advantages, challenges, and future prospects of its implementation in the Industrial Internet of Things.

OPC UA, or Open Platform Communications Unified Architecture, is a complete distributed system that enables interoperability and communication between industrial automation devices and systems. It provides a standardized communication protocol for exchanging data and information between machines, devices, and applications from different vendors. It is a service-oriented architecture built around a client-server model, enabling it to be used in various applications such as industrial and building automation and energy management.

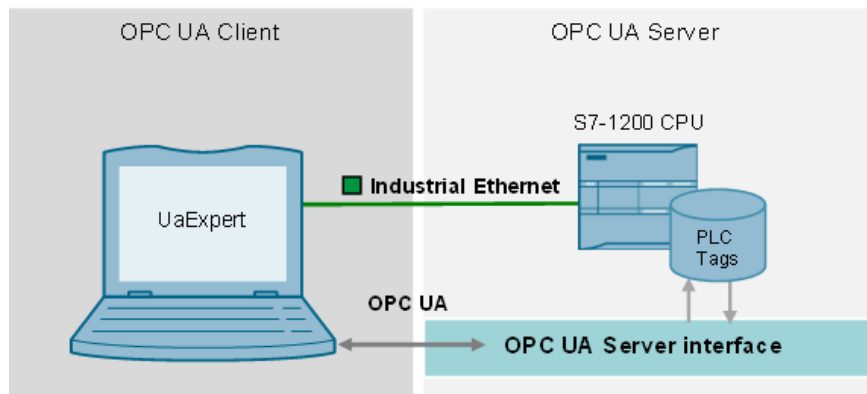


Figure 4. OPC UA Server Interface. (Source: <https://www.kb-controls.io/post/plc-plc-communication-with-opc-ua>)

OPC UA's key feature is its ability to abstract and model data, devices, and systems in a vendor-neutral way, allowing for interoperability and consistency between different vendors. In other words, it offers a standardized way of describing and modeling these elements that can be used across different platforms and systems, without the need for proprietary or manufacturer-specific protocols. This feature promotes interoperability and compatibility among different systems, making it easier to integrate diverse devices and technologies in a unified framework. It also supports various security features such as encryption and authentication, making it essential for securing industrial systems and ensuring the integrity of the data. In many cases, OPC UA is used to exchange data and information between different Industrial Ethernet protocols, allowing different devices and systems to communicate with each other even if they use different communication protocols.

OPC UA enables efficient and accurate control of the additive manufacturing process through seamless data exchange between different devices and systems. It allows for the creation of a fully integrated and connected system where devices and machines can share data and information with each other in real-time, leading to improved efficiency, accuracy, and productivity. OPC UA provides a profile mechanism for the description, classification and discovery of implementation features. The mechanism allows communication between different devices like programmable logic controllers (PLC) and powerful server machines. The use of OPC UA in M2M Machine to machine (M2M) communication is becoming increasingly important in the era of Industry 4.0 (Drahos et al., 2018), where the focus is on creating smart factories and systems that are interconnected and can communicate with each other in real-time. The adoption of a distributed communication system enables the creation of a comprehensive, interconnected network of devices and machines, which facilitates the exchange of data and

information. This, in turn, can lead to improved levels of efficiency, accuracy, and productivity. Over time, various studies have examined the use of OPC UA in comparable settings, as evidenced in studies such as those conducted by Friihwirth et al. (2015) and Schiekofer et al. (2018). This last study 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. For a while now, OPC UA has been regarded as a crucial communication protocol in the industrial automation field (Derhamy et al., 2017), and it is considered a key enabler of Industry 4.0, as stated in Bauer et al. (2021). 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 open62541¹, which is a dedicated open-source software repository for OPC UA in general. This library allows for reading the current value of a node and monitoring its values. The original library was implemented in C and could present challenges in certain scenarios. In the 3YOURMIND implementation, it has been encapsulated within C++ code to enhance its usability. As an example, in the 3YOURMIND Machine Connectivity implementation this values can monitored through a callback function referred to as *"LoggingNodeValueAssigner"*, the value of the node is subsequently stored in an appropriate variable (e.g., *"Polyline::CurrentNodeValue robot_cell_handling_robot_ready"*), that then can be monitored by the workflow controller . In the Polyline project a preparatory measure for 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 displayed in the blackboard entry in the Behavior Tree.

As a hint into the real application of OPC UA communication it is important to highlight as well 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, and users must ensure that the new port is properly documented and shared with all relevant parties to maintain communication compatibility.

Within the framework of this research, it is worth dicussing the structure and functioning of some of the OPC UA implementations by the involved manufacturers. Several companies, including DyeMansion and Krumm-tec, have

¹ <https://www.open62541.org/>



opted to use Siemens hardware and software in the development of their machines. The utilization of Siemens STEP 7 Professional application has facilitated the provision of an OPC UA server for these machines. This is an illustration of the advantages and ease of integration offered by the Siemens platform when implementing OPC UA-based solutions across various applications. Additionally, Siemens has developed an extensive platform 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 (STEP 7 professional (TIA portal), 2023), as well as OPC UA server and client libraries. This platform offers native support for OPC UA, making it an attractive choice for engineers and developers seeking to implement OPC UA-based solutions in industrial automation and other domains.

2.3.1.1 UMATI

UMATI (Universal Machine Type Interface) 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 aextensible standard that supports complex information modeling that offers reliable communication between machines and controllers in industrial environments (Friihwirth et al., 2015) 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. In the context of the IIoT, UMATI is an important building block for enabling communication and data exchange between devices and systems in an industrial environment.

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 (Maia et al., 2020)

and (Drahos et al., 2018). Once the UMATI standard is fully defined, it will simplify the process of connecting disparate machine types and applications. It will also streamline the linking of machines from multiple vendors and enable easy integration with other OPC UA implementations, even if they have not yet implemented UMATI.

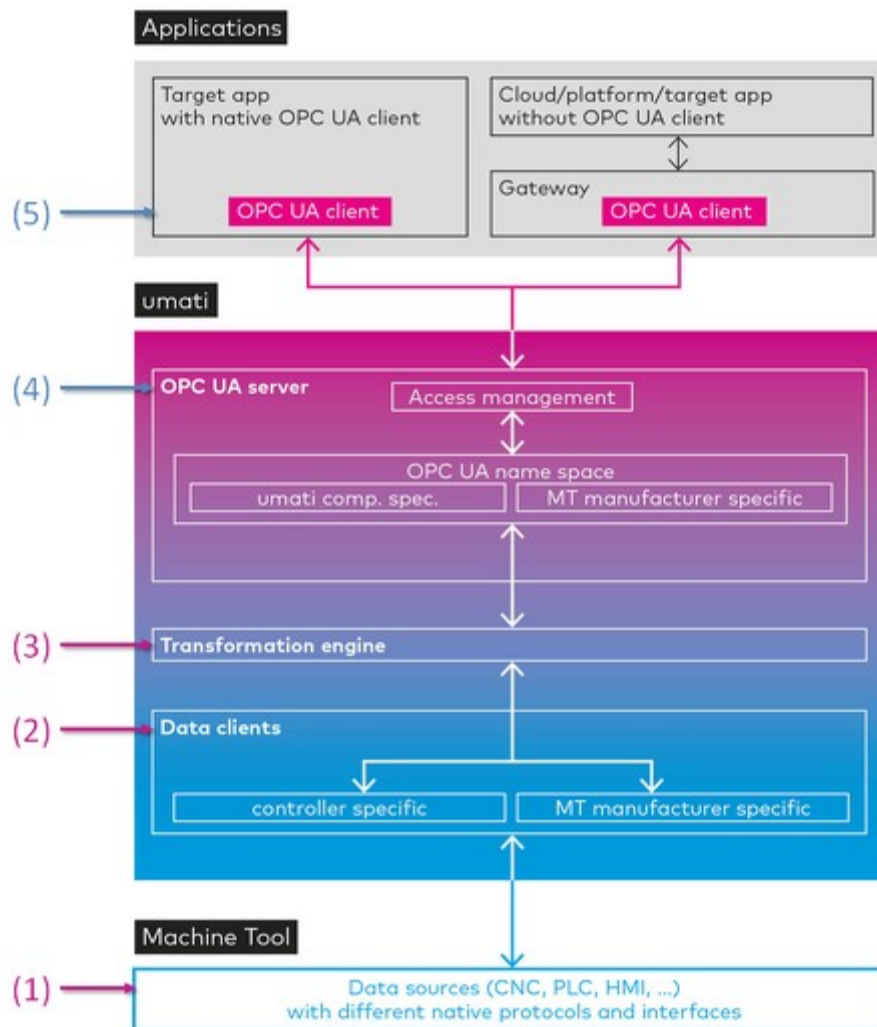


Figure 5. Configuration required to make a machine tool compatible with UMATI (Source: Edition based on the diagram specifications published in VDW)

Figure 5 outlines the configuration required to make a machine tool compatible with UMATI. The operating information is stored in the control device and software (Data sources (1)). The data clients (2) retrieve the operating information, and 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 comprise the UMATI gateway, which connects machine tools to UMATI. The OPC UA client (5) connects systems and software applications to the UMATI gateway via a network. The system allows machine tools to be accessed using the same UMATI data format, regardless of the manufacturer or model of the connection destination, making it possible to acquire operation information.

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. While 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 schema utilized is inspired by the UMATI schema, and the data sent to the MES follows the same structure as the UMATI schema. The information is serialized under the file *umati.proto*, using the proto buf format, which is an encoding format developed by Google that will be better explain in Section 3.2.

2.3.2MQTT

MQTT is a lightweight message protocol that uses a subscription-publishing model to allow publishers to send messages to a server that forwards them to subscribers, avoiding point-to-point connections between subscribers and publishers. It is designed for implementation in low-resource devices, low-bandwidth networks, and high latency. The protocol defines two entities in the network, as shown in Figure 6: the broker (server) and clients. Topics are the routes where messages are published within a broker, allowing information to be sorted and messages to be better managed. The message can be in any data format for the payload, such as JSON, XML, encrypted binary or Base64, and supports three different levels of quality of service for sending messages. MQTT is ideal for low-power devices that need to send information to a server, and its low bandwidth consumption and flexibility make it suitable for a variety of devices, including Arduinos, Raspberry Pis, and commercial home automation solutions. Verma, Prashant. (2022).

MQTT (Message Queuing Telemetry Transport) is a publish/subscribe-based messaging protocol designed for use in low-bandwidth, high-latency, or unreliable network environments. It is commonly used for Internet of Things applications where devices and sensors need to send and receive data over a network. MQTT provides high performance and low latency for IoT and M2M (Machine-to-Machine) communication, due to its lightweight and efficient design.

MQTT is ideal for low-power devices that need to send information to a server, and its low bandwidth consumption and flexibility make it suitable for a variety of devices, including Arduinos, Raspberry Pis, and commercial home automation solutions (Azzedin & Alhazmi, 2023). The protocol's use of binary encoding and efficient compression techniques helps to minimize its overhead and improve its performance, while the support for different levels of Quality of Service

(QoS) for messages allows clients to control the level of reliability they require for their messages.

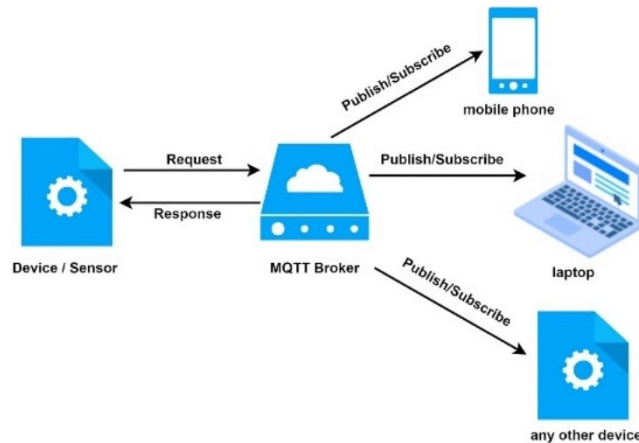


Figure 6, MQTT schema. Source: <https://www.sciencedirect.com/science/article/pii/S2352864822000347>

The ISO/IEC 20922 standard specifies that MQTT can handle data rates up to 256 Mb in size. MQTT is composed of three fundamental components: a subscriber, a publisher, and a broker. In MQTT, there are three fundamental components: a subscriber, a publisher, and a broker. A publisher is a device or application that sends messages to a broker, which acts as an intermediary between publishers and subscribers. Publishers send messages to the broker on a particular topic, which acts as a channel for communication. A subscriber is a device or application that receives messages from the broker. Subscribers subscribe to topics of interest and receive messages that are published on those topics. 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. The broker manages the connections between publishers and subscribers, as well as the topics to which they are subscribed. It is also responsible for enforcing security and access control policies, and for managing message queues when subscribers are not available. Together, these three components form the basic building blocks of MQTT, enabling devices and applications to communicate with each other in a lightweight and efficient manner.

2.3.3 Http

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 leveraged 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 leverage HTTP to transmit and receive data. The REST API is an architectural pattern for building web services that adheres to the principles of Representational State Transfer. These principles aim to create scalable and flexible web services that can be easily consumed by a broad range of clients, including web browsers, mobile devices, and other applications.

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 was tested using Markforged 3D printers. These printers are equipped with the Eiger API V3 (Eiger API V3, n.d.), which operates on the REST API architecture. As will be further elaborated upon later on, the testing involved 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 printing and monitoring the print progress.

It is important to understand the distinction between RESTful and REST API. A RESTful API is an API that adheres to the principles of REST. It is a web service that follows the REST architectural style and provides a set of operations that can be performed on resources identified by URIs. A REST API, on the other hand, is simply an API that uses REST principles. 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, (Fielding, 2000). REST is a popular method for connecting components and applications within a microservices architecture due to its flexibility and autonomy for developers. REST API is a way of interacting with web and cloud services using HTTP requests, as described by Schiekhofer et al. (2018). 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.

Talking about the main characteristics of REST we can say that it's characterized by its uniform interface, with its four constraints of resource identification, resource manipulation, self-descriptive requests and responses, and hypermedia as the engine of application state. REST is also based on a layered system, where each layer provides services to the layer above and uses services from the layer below. The client-server interaction in REST is stateless, meaning no session state is allowed and all necessary information must be included in each message. In the context of REST APIs, statelessness means that each request made to the API contains all of the information necessary to complete the request, and the server does not store any client context or session information between requests. This means that the server does not have any memory of previous requests, and each request is treated as an isolated transaction.

This design principle is in contrast to stateful systems, where the server maintains a session state, and each request is dependent on the state maintained by previous requests. The stateless design of REST APIs makes them more scalable, cacheable, and easier to maintain, as there is no need to store session data on the server. Additionally, it also allows for greater flexibility, as the client can make requests to any server at any time, without having to worry about maintaining a session or maintaining state information on the server.

There has been a significant amount of research and technological progress in the implementation of REST APIs in various contexts. One example is the use of REST APIs in Android mobile apps, as studied by Abdellatif et al. (2020).



Another example is the use of REST APIs in cloud testing, as explored by Wolde & Boltana (2021). REST APIs are widely utilized in a variety of applications, including web applications, mobile apps, IoT devices, and the Industrial Internet of Things (IIoT). They are employed for tasks such as data retrieval, creation and updates, user authentication, and access control to resources. 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. In the context of the IIoT, REST APIs play a crucial role in enabling communication and data exchange between devices and systems over a network. This is due to the simplicity, flexibility, and scalability of REST APIs. By using REST APIs, devices and systems can effectively communicate and exchange data, allowing for greater efficiency and effectiveness in the IIoT.

The performance of a REST API implementation can vary depending on several factors, including the complexity of the operations being performed, the efficiency of the server-side code, the speed of the network and the devices being used, and the number of concurrent users accessing the API. In general, REST APIs are designed to be lightweight and fast, as they use the HTTP protocol and rely on the client-server model.

3 Methodology and results

The present study will be conducted within the framework of Machine Connectivity 3D Pulse Control (3YOURMIND, 2023). This IIOT solution is based on the utilization of the OPC UA communication standard and the UMATI.

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 create a trigger utility that will start the workflow controller, this trigger utility will also be used for pulling production reports from a Markforged machine based using Representational state transfer (REST) a standard for internet data exchange this data transfer with a real machine is a key component 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 (as said before based on Behavior Trees) implementing the machines belonging to the production line of the project. The software development approach chosen in this research is Test Driven Development through automated mock testing with GoogleTest (also known as gtest) and Doctest frameworks in C++. This approach has proven successful over time and is now considered to be a good practice (Petrovic et al.,2021)

To ensure the achievement of the determined objectives in this research the implementations will be tested and simulated. In addition, the complete POLYLINE project will be integrated and tested in the BMW production structures. Simulation was chosen as the primary testing method. However, in Section 3.6 the industrial testing process will be discussed to some extent, as well as highlighting some of the tested advantages of Behavior Trees, the data collected from the on-site results of this testing will be analyzed in order to derive Key Performance Indicators that provide a general understanding of the performance trends in this implementation.

3.1 Project outline

Before embarking on a discussion of the various implementations and testing that have been conducted, it is imperative to first gain a thorough 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 7, provides a comprehensive illustration of the operations that take place within the framework of Machine Connectivity. It is through this representation that the underlying design principles, communication channels, and component interactions can be fully comprehended.

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 signal aggregator receives input signals which serve as stimuli for the activation of Behavior Trees.

Upon activation, the Behavior Trees then execute their nodes by issuing specific command instructions to the designated drivers of the machines, thereby facilitating the execution of the intended actions. The Behavior Tree dispatches commands to the various machines (including Falcon, Grenzebach, EOS,...). Upon receipt of the commands, the machines execute the instructions and send the results back to the Behavior Tree, which reacts accordingly. The signal aggregator then forwards the results to the Manufacturing Execution System.

Upon observing Figure 7, it is pertinent to clarify the role and functionality of the "drivers," despite their exclusion from this thesis's scope. 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.

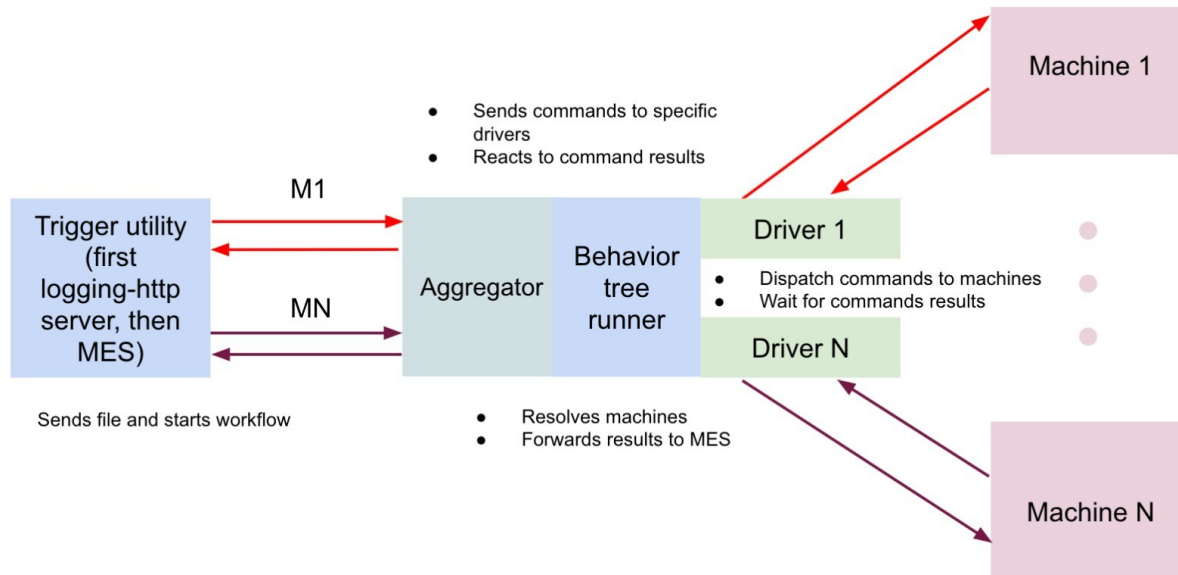


Figure 7. Schema of the connections and functionalities in the project.

3.2 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 various 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 (MES) and industrial machines, and to identify areas for improvement. The trigger utility was envisioned as the 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, when the correct Behavior Tree node would have been triggered, to send 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 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 and features interactive documentation generated from the specification file, making it easy to understand and use. OpenAPI Specification (OAS) is a standard for describing and documenting RESTful APIs. It is used to define the structure of an API, including its endpoints, operations, parameters, and responses. The Eiger API is structured around resources and collections of resources that are accessible through unique URIs. Interaction with these resources is performed using standard HTTP methods. The Eiger API primarily uses JSON to exchange data between client and server,

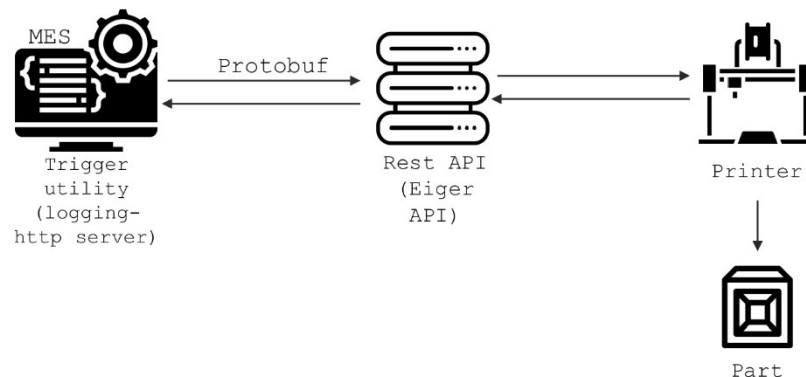


Figure 8. Communication architecture of the implementation with the Markforged printer

Even though the Eiger API primarily uses JSON to exchange data between client and server it possible to use Protobuf (Protocol Buffers), a binary serialization format developed by Google, to exchange data with the Manufacturing Execution System. Protocol Buffers are a data serialization format that is language- and platform-agnostic, making them well-suited for use in REST APIs like the Eiger API. 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 (Kumar, 2019), also shown Figure 9, in this study, the difference in time performance is minimal owing to the small size of the transmitted message. However, as the message size increases, the cost of unmarshalling the message into JSON format increases correspondingly.

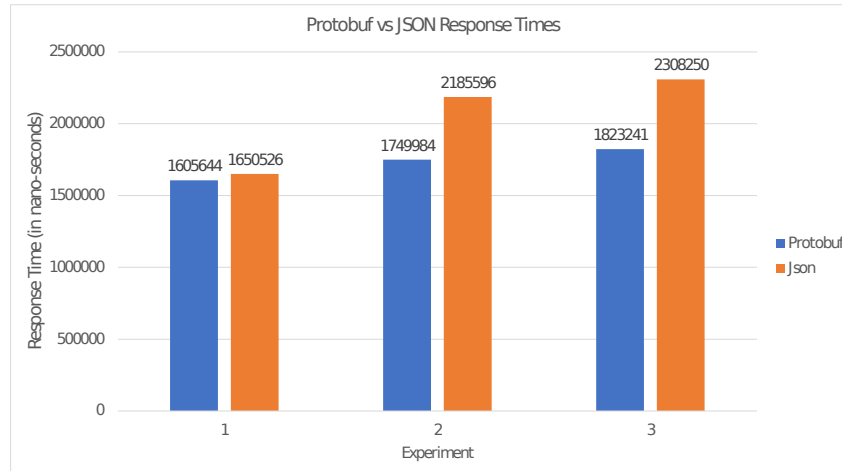


Figure 9. JSON vs Protobuf performance comparison

In the context of the Eiger API, the use of Protocol Buffers enables the efficient exchange of information between the API and the Manufacturing Execution System, making it easier to automate and streamline the 3D printing process. Access to the Eiger API is secured through authentication, which is used to identify the API consumer and verify their authorization to use the API. The authentication process is accomplished through HTTP Basic Authentication and the use of API keys. API keys for Eiger consist of two components: an API Access Key and an API Secret Key.

The illustration below 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 utilized was done using an open-source tool². It is noteworthy to emphasize that this graph only encompasses a fraction of the .proto file, as constructing a graph with all the implemented numerous 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.

² <https://github.com/seamia/protodot>

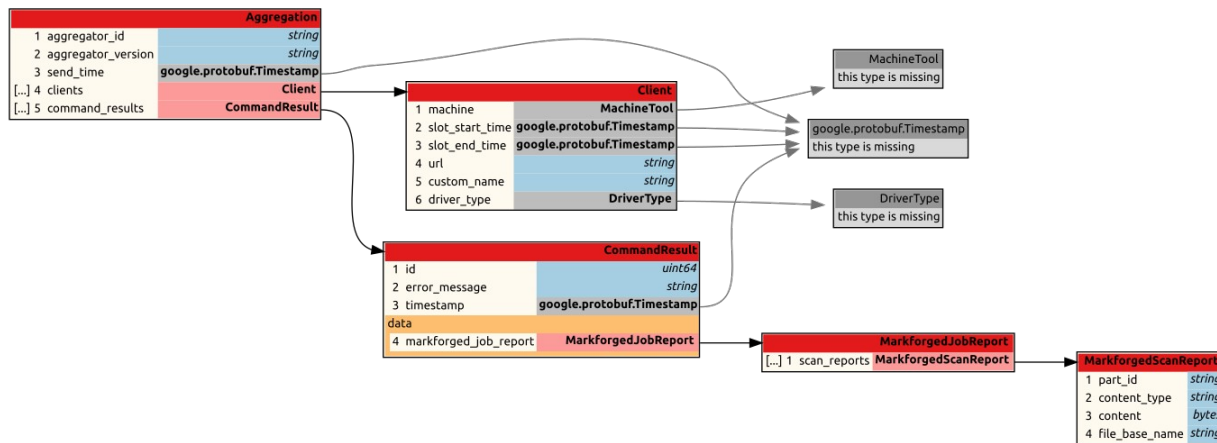


Figure 10. Schema of the Markforged communication

This implementation was put to the test and demonstrated satisfactory outcomes. It facilitated the activation of a designated machine, and also enabled the generation of a PDF document containing information on both the print process and communication data. The successful results of the testing indicate that this implementation is effective in achieving its intended triggering purpose. The ability to initiate a machine and generate a comprehensive record of the print and communication data in a portable and easily accessible format could enhance the overall efficiency and effectiveness of the workflow.

The development of the utility underwent testing using the company's tools. However, the initiation of the process remotely 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. The outcome of the testing phase using the company's tools provided valuable insights for future considerations and advancements. However, the implementation of remote starting is still subject to the resolution of regulatory and policy restrictions.

3.3 Comparison between protocols

In order to contrast the various communication protocols examined within the scope of this study, the research conducted by Chris in 2019 will serve as the foundational point of reference.

In a comparable study (Dizdarević et al., 2019) 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 (IoT). The researchers ultimately found that RESTful HTTP and MQTT are the most advanced protocols, with MQTT demonstrating exceptional performance on resource-constrained devices. They also determined that the performance of RESTful HTTP is inadequate for integrating IoT, fog, and cloud computing solutions.

OPC UA is an industrial communication standard for machine-to-machine or industrial Internet of Things communication. It has several advantages over MQTT, one of which is its ability to connect machines to each other. OPC UA provides a secure and reliable means of exchanging data between machines, allowing for real-time data exchange with a high degree of semantic interoperability. This means that data can be easily and accurately interpreted by different systems, even if they have different underlying technologies or protocols. Also provides a hierarchical structure for data representation, which makes it easy to manage and organize large amounts of data generated by multiple machines. It also supports multiple data types, including complex structures and arrays, which makes it suitable for a wide range of industrial applications.

In contrast, MQTT is primarily designed for small-scale, low-bandwidth IoT devices and provides only a basic publish/subscribe messaging model. It does not provide the level of interoperability or data management capabilities that OPC UA offers, making it less suitable for industrial M2M communication.

In summary, the capacity of OPC UA to establish interconnectivity among machines, in tandem with its dependable and secure data transmission, sophisticated data depiction, and semantic interoperation, positions it as a superior alternative for industrial machine-to-machine communication, as opposed to MQTT.

Additionally, OPC UA's support for complex data types and ability to manage and organize large amounts of data make it well-suited for industrial applications with diverse data requirements. In contrast, MQTT's limited messaging model and bandwidth capacity restrict its functionality in industrial settings.

REST API, while flexible and suitable for web-based communication, lacks the security and real-time capabilities of OPC UA, making it less suitable for industrial IoT communication. Overall, OPC UA's advanced features and secure data exchange make it the optimal choice for industrial IoT communication, while MQTT and REST API remain viable options for simpler applications.

3.4 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 (Faconti, 2022). It is worth noting that the compatibility of Groot 1.0 is exclusively

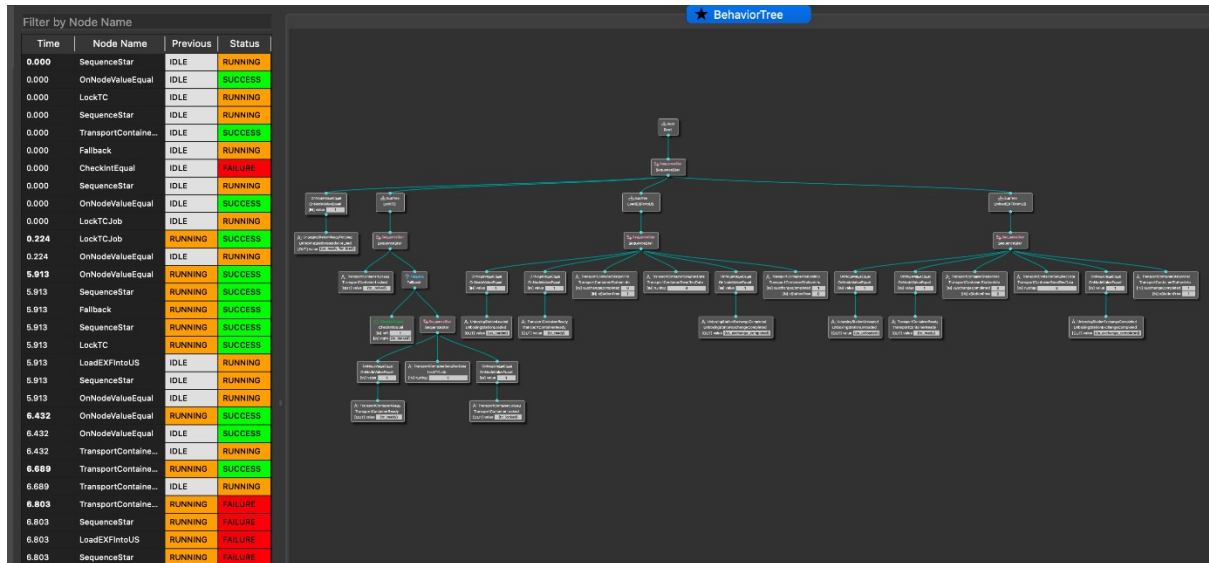
Figure 11. Log replay from the testing of one subtree.

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, (Faconti, 2019) 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, thus 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 11, 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.

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.

Moreover, as illustrated in Figure 12 and expounded upon by Ghzouli et al. (2022), 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.

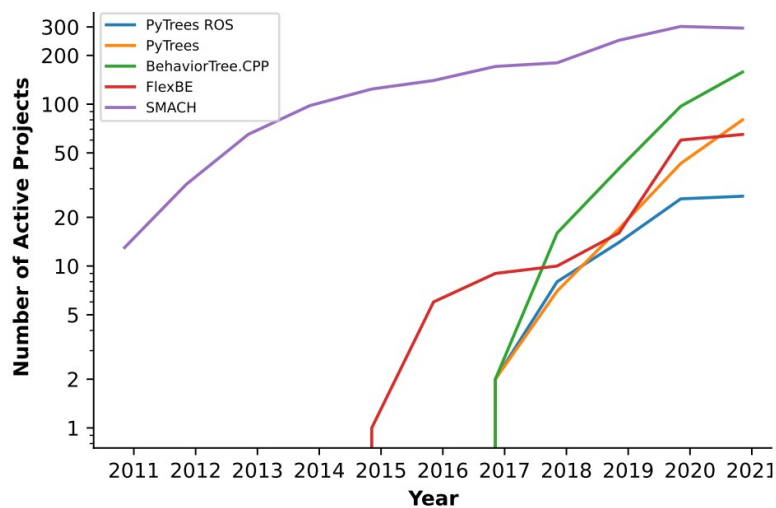


Figure 12. A study of the trend of usage of state machine and behavior tree languages in open-source projects on GitHub over time. Extracted from (Ghzouli et al., 2022)

3.4.1 Workflow design

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

The image depicted on **Figure 13** 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, comprising 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) functions as a post-processing apparatus for the printed part. The Grenzebach RobotCell, which encompasses the handling and bin-picking components, enabling the transportation and classification 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, signifying the 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. As such, the AGVs cannot load the machine automatically. Currently, the part heating process is occurring directly on the printer.

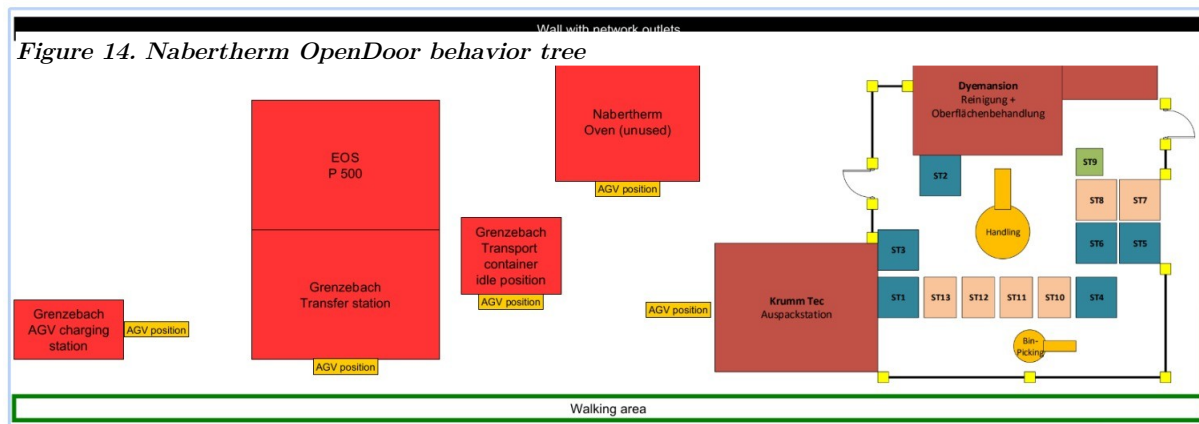
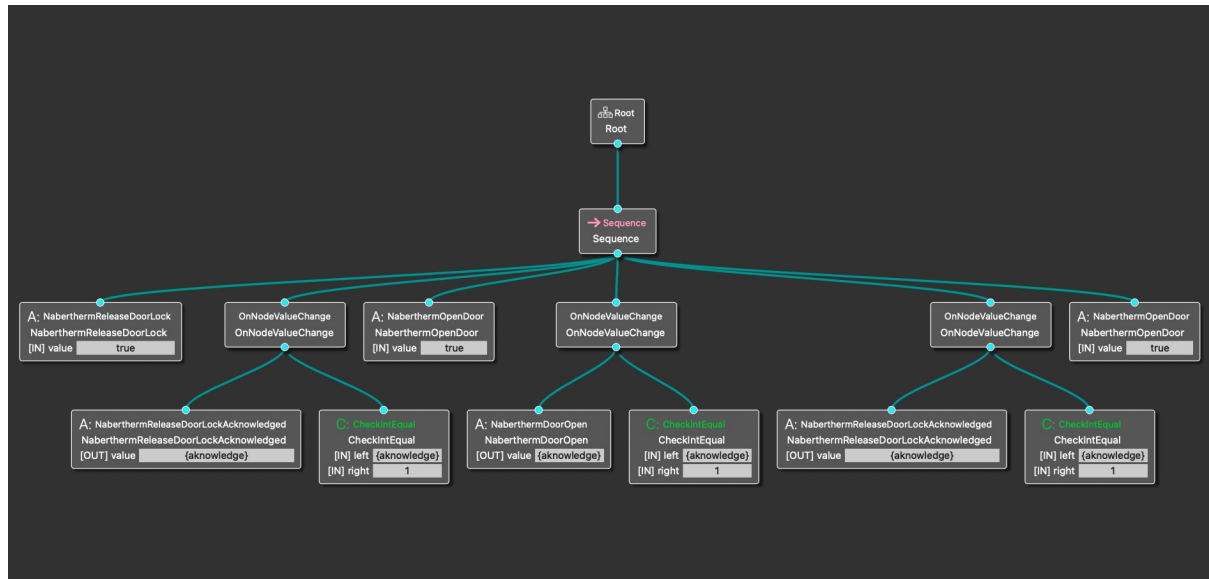


Figure 13. Workflow schema

Prior to delving into the specific details of the processes involved, an exposition of the logic process 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, based on the simple nature of its signaling and ease of interpretation. To illustrate this point, the photograph below showcases the Nabertherm furnace as an exemplar. It is noteworthy that during the design phase of this research and the signaling process for the entire project, 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,

close the Nabertherm Door and Start and Stop the heating, as shown in Figure 14, they originally would have become subtrees of the final tree.



However, as the construction of the decision trees progressed, it became evident that this approach was not the best course of action. The lack of collaboration between different processes and machines resulted in a limited scope of operations and hindered 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. Therefore, it was decided to construct the decision tree based on specific operations that involve the integration of various machines from multiple companies. This approach allowed for a comprehensive analysis of the operations and allowed for a more in-depth examination of the processes involved. The revised approach resulted in a more comprehensive and effective decision tree that improved the efficiency and accuracy of the operations analysis. The integration of multiple processes and machines allowed for a more holistic approach that resulted in a better understanding of the relationships between the different operations and their impact on the overall outcome. Nevertheless, as it will be discussed in Section 3.5, this approach allowed to initially test the mock environment simulation viability.

After this testing the required workflow was implemented and divided in different sections. The subsequent three sections, that can be seen in Figure 15 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. Despite being executed sequentially, the final behavior tree file [*tree.xml*] is divided in subtrees due to its sheer size, 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 7, this four-sectioned structure constitutes the primary behavior tree and represents the main logic behind the workflow.

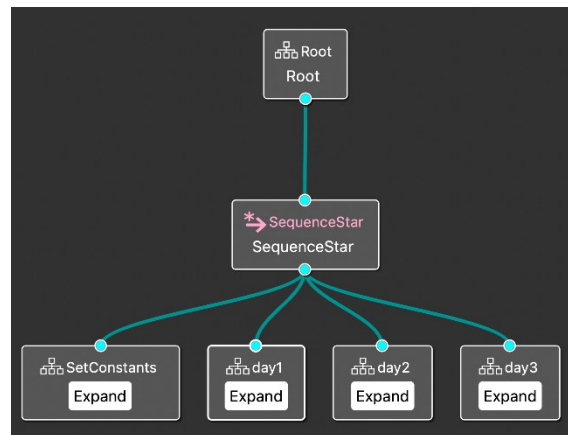


Figure 15. Main behavior tree of the project

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 the "SetConstraints." This subtree serves as a foundational step to establish the parameters required for the overall workflow. The first step of this



tree involves the use of the "SetBlackboard" node, which is part of the BehaviorTree.cpp library. The function of this node is to set the values of signals utilized for communication between the tree and the machines.

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. It enables nodes in the tree to read and write data, which can be used to make decisions and control the system's behavior. 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. The SetBlackboard node is particularly useful for storing and accessing data that needs to be shared across different parts of the Behavior Tree.

It is important to note that this initial phase of 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 aim is to provide a user-friendly and intuitive interface for setting the parameters of the workflow.

The division of the workflow into different days is an essential aspect of the process, as it enables the effective and efficient execution of the production process. The workflow division ensures that each day is devoted to specific tasks that can be completed within a day, making the process more manageable and organized. On the first day, the process involves the collaboration of three different partners, which include the EOS P500 3D printer, the Grenzebach Transfer Station, and the Grenzebach Fleet Management System (AGVs). These partners are responsible for specific tasks in the production process, and their collaboration ensures that the tasks are completed seamlessly. The second day of the workflow is focused on the Grenzebach Transfer Station and the Krumm-tec Unpacking Station. This day's tasks are interrelated, and the collaboration between these partners ensures the smooth completion of the tasks. The third day of the workflow involves the Krumm-tec Unpacking Station, the Falcon DyeMansion, and the Grenzebach RobotCell. These partners are responsible for specific tasks, and their collaboration ensures that the production process is executed effectively and efficiently.

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 transportation container 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. Shown in Figure X

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). At the conclusion of the day, the workflow controller should have

issued all the requisite 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 assesses whether the necessary precondition above mention have been satisfied and whether the serialization of the Additive Marking is underway. 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 and is outside the scope of this study. After cheking 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, as part of the progression from X1Y1Z1 to X2Y1Z1, better despiected in figure X. 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, and the exchange frame is positioned already in the designated location with the completed print component, where it will undergo a cooling process.

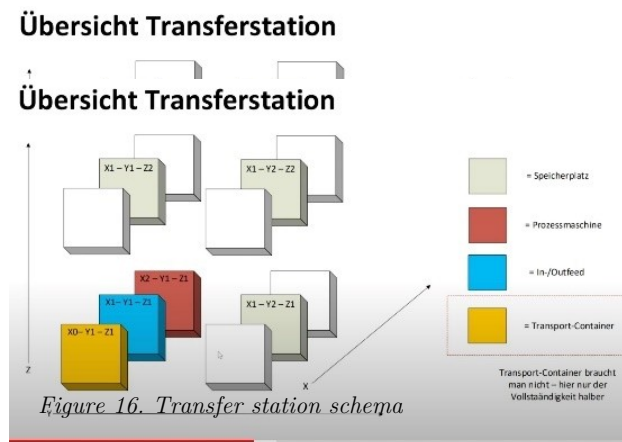


Figure 17. Transfer station schema

On the second day of the workflow, once the printed part is within the printer and cooled down, the printed part that was previously situated within the EOS P500 3D printer is subsequently transferred to the designated transport container. The transport container is then moved to the Krumm-tec Unpacking Station for further processing.



The Day 2 workflow is contingent upon certain 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 X. Upon verifying the aforementioned conditions, the exchange frame is relocated to the InOutFeed station, signifying a transition from the red box to the blue box in Figure X. Notably, any exchange frame intending to exit the transfer station must first proceed to the InOutFeed station, from which it will be retrieved by the Transport Container. Upon the relocation of the exchange frame to the InOutFeed station, the workflow initiates a command to the Transport Container, instructing it to secure itself into the appropriate position. 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, causing it 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 which point the fleet management system is notified to dispatch one of its vehicles to retrieve the Transport Container and convey it to the designated unboxing station (US). The fleet management system handles the requisite 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 X, which is designated as the location for the installation of the printed component. The Unboxing Station initially contains the Larged Empty Dirty (LED) box, which serves as the designated storage location for the printed component. Once the printing process is concluded, a large box containing powder is acquired, which must be emptied by the unboxing station to isolate the printed component. The powder residue, which soils the box, renders it designated as Large Full Dirty (LFD) at the completion of the workflow.

The workflow on Day 3 commences with the presence of the printed box and its constituent parts located in Station 1. A prerequisite for the initiation of the workflow is the requirement that the printed box be present in Station 1. The objective is to have all of the individual components separated and prepared for removal by the end of the day. This is to ensure that the workflow is successfully concluded (in Station 7), and that the components are readily available for use outside the system. The ultimate goal of the workflow is to have all of the individual components arranged in their designated boxes at Station 7, thereby rendering them available for removal from the system.

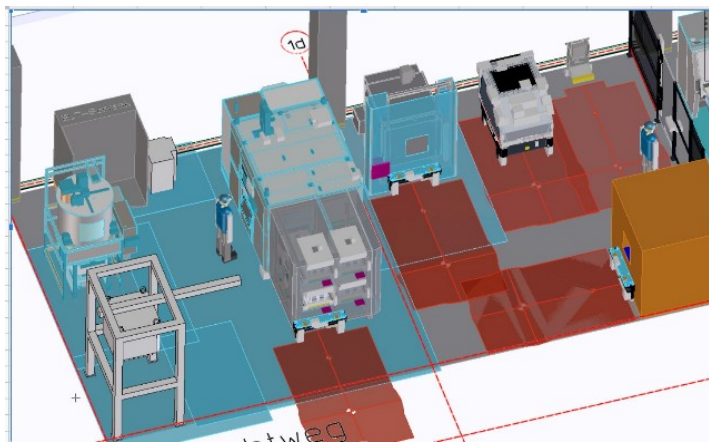


Figure 18. Schema of the workflow

The Day3 subtree commences by directing the relocation of the printed component box to the Falcon, and instructs the robot cell to move from Station1 to Station 2.1 (which is utilized for accommodating full boxes at the Falcon, and will receive the cleaned parts at the conclusion of the process). In addition, an empty large box is relocated to the Falcon and positioned at 2.1, serving as the receptacle for the clean parts. The transfer of dirty boxes is limited to Stations 1 and 1, whereas the clean boxes can be relocated to any other station, to ensure that there is no commingling between the clean and dirty boxes. Clean boxes are initially sourced from Station 3, which serves as a buffer, and must be available as a precondition. Although it is possible for clean boxes to be fed into Stations 5 and 6 by personnel, this is not part of the workflow, 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's 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. The dirty box is relocated from 2.3



(which corresponds to the Larged Empty Dirty Box) at this stage. 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 allocated to 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 quantity of distinct parts that will be generated, which will determine the Sepoint (i.e., the expected number of parts). Subsequently, the bin picking operation is performed, with the robot ceasing operation upon completion. 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 out feed (i.e., the initial loop of Day 3), where individuals can retrieve them. However, the additional precondition specifies that space must be available at the out feed 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 bin picking stations utilized, there may be up to four boxes required. Thus, the full boxes are transferred from Stations 10-13 to Station 8, while empty boxes are relocated from Station 7 back to Stations 10-13.

3.4.2 Machines and Behavior Tree Nodes

This section provides a comprehensive summary of the implemented behavior nodes and their interconnections with the manufacturers and machines involved in the process. The purpose of this section is to encompass all of the nodes devised by 3YOURMIND and to provide a coherent representation of the associated research.

Manufacturer	Machine	Nodes
--------------	---------	-------



DyeMansion	Falcon	DMFalconAbortJob DMFalconLoadStateId DMFalconReady DMFalconSetJob DMFalconStart DMFalconUnloadStateId
	DM60	-
Krumm-tec	Unpacking Station	UnboxingStationBusy UnboxingStationError UnboxingStationExchangeCompleted UnboxingStationLoaded UnboxingStationReady UnboxingStationReadyForLoad UnboxingStationForUnload UnboxingStationUnloaded
Nabertherm	Oven	NaberthermCloseDoor NaberthermDoorClosed NaberthermDoorOpen NaberthermOpenDoor NaberthermReleaseDoorLock NaberthermReleaseDoorLockAcknowledged NaberthermStartProgram NaberthermStopProgram
EOS	EOS P500	EosExchangeFrameLoadCommandState EosExchangeFrameUnloadCommandExecute EosExchangeFrameUnloadCommandState EosJobStartCommandExecute EosJobStartCommandState



	EosPauseCommandExecute
TransportContainer	TransportContainerBusy TransportContainerExchangeCompleted TransportContainerLoaded TransportContainerLocked TransportContainerReady TransportContainerSendRecData TransportContainerStationInfo TransportContainerUnloaded TransportContainerUnlocked
TransferStation	TransferStationBufferEXFReady TransferStationDoorOpen TransferStationExchangeCompleted TransferStationInOutFeedEXFReady TransferStationInOutFeedEmptyEXF TransferStationInOutFeedFree TransferStationJob TransferStationOperateDoor TransferStationPrinterEXFReady TransferStationPrinterFree TransferStationREadyForJob TransferStationStatusInOutFeed
FleetManager (AGV)	FleetManagerGetTaskStatus FleetManagerTriggerPickAndDrop
RobotCell	RobotCellActualPartsAtStation11 RobotCellActualPartsAtStation12 RobotCellActualPartsAtStation13 RobotCellActualPartsAtStation14 RobotCellBinPickingRobotReady RobotCellChangeProcessState RobotCellHandlingRobotReady

RobotCellHandlingState
RobotCellInFeedSmallKLTInPos
RobotCellOutFeedSmallKLTInPos
RobotCellSendOrder
RobotCellSendSetpointsForBinPicking
RobotCellUnboxingStationKLTInPos

3.4.3 Communication nodes

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 pertain to the OPCua communication protocol, which was selected by the machine manufacturers involved in the workflow. The OPCua 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 are the Blackboards, as explained in [Section X](#), 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 simultaneously. This allows the nodes to coordinate and collaborate with one another, enabling the Behavior Tree to make more 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 used to receive information from the environment or other nodes in the tree. They provide the necessary data for a node to carry out its designated task. When an input node is ticked it stores the value of the designated variable on a blackboard key that can then be used by the entire rest of the tree. Output ports, on the other hand, are used to send information from a node back to the environment or to other 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 sections before we have discussed the OPCUA connection logic and even if this connection was not directly part of this research process it is vital to explain for the comprehension of the log of the node. Machine Connectivity saves the values of the nodes from the OPCUA server from each of the machines, when for example and output behavior tree node is ticked the value of the opcua node from the server gets put onto the blackboard. In other words, when certain output behavior tree node gets ticked it copies the current value of the node in a thread-safe manner and sets it into the output port, we can be typically be remapped to a blackboard key by using the {} syntax. so for example in `<Action ID="TransferStationPrinterFree" value="{ts_ready_printer_free}"/>` it would be that `"{ts_ready_printer_free}"` saves the value of `value_nodes.transfer_station_printer_free`, so the key `"{ts_ready_printer_free}"` can be then saved and analysed by the tree. When we are talking about the opposite case and we have an input value on the behavior tree node that is ticked, the code takes the value from the input port (remapped to a blackboard entry and executes and asynchronous action that perform a node-value.

To comprehensively explicate the nodes within the Behavior Tree, it is imperative to first understand the connection established between the machines, as elucidated in previous sections of this study. While this connection was not a direct focus of this research process, it is crucial to contextualize the log of the nodes.

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.

For example, in the XML syntax from the tree `<Action ID="TransferStationPrinterFree" value="{ts_ready_printer_free}"/>`, the key `"{ts_ready_printer_free}"` saves the value of the OPCua node represented by `value_nodes.transfer_station_printer_free`, thereby allowing the key

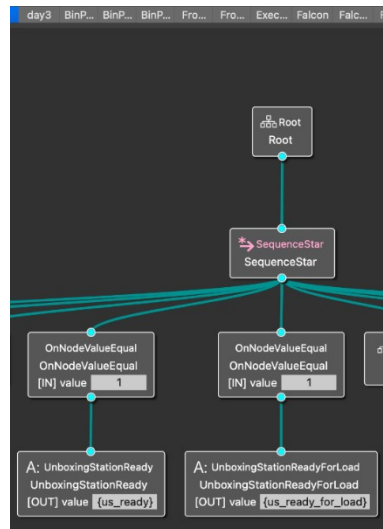


Figure 19. Subtree representation day 2 of output nodes

"{ts_ready_printer_free}" to be saved and analyzed by the tree. The *value_nodes.transfer_station_printer_free* saves the OPCua value that shows the status of the printer.

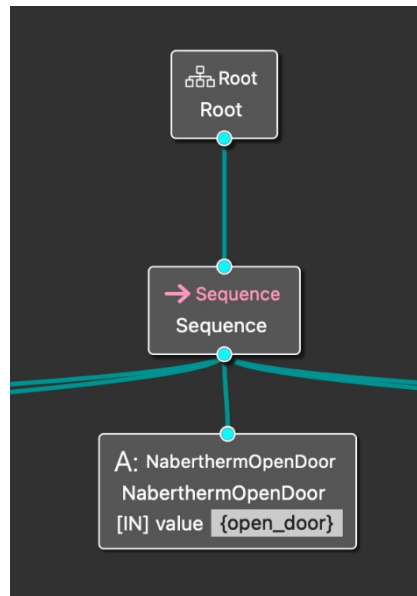
Figure 20. Subtree representation, example of input nodes

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 that performs a node-value operation.

Regarding Grezenburg, for example, the manufacturer of the Transfer Station, the OPCUA nodes utilized in this study were obtained 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.

The tables and pictures below showcase the implementation explained

BLACKBOARD		
Key	Type	Value Node
{us_ready}	Boolean	unboxing_sation_ready
{us_ready_for_load}	Boolean	



BLACKBOARD		
Key	Type	Value
{falcon_job_id}	16bit integer	
{falcon_pre_process}	boolean	True/False
{falcon_program_number}	32bit integer	
{open_door}	boolean	nabertherm_open_door

3.4.4 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 controlling the flow of execution



within the tree. They serve to organize and group other nodes within the tree, enabling the direction of behavior.

The control nodes utilized in this workflow include both traditional composite nodes, such as Sequence and Fallback, and specialized control nodes like SequenceStar, OnNodeValueChange, and OnNodeValueEqual. These specialized nodes were selected for their unique ability to effectively address the specific challenges posed by the workflow, and to improve the overall functionality and performance of the system.

The Sequence and SequenceStar nodes are both used in behavior trees to execute child nodes in a specific order, but they have opposite behavior. The Sequence node starts executing its child nodes from the beginning each time it is called, 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 Behavior Tree library. This distinction is important to understand when designing and implementing behavior trees, as it can have a significant impact on the behavior of the system.

Apart from the common nodes already explained in section X, we have to also highlight some of the nodes from the BehaviorTree.cpp library most used, in them we find the SequenceStar node, seen in Figure X.

The usage of Sequence and Fallback control flow nodes in Behavior Trees may require frequent checks to determine if a child node needs to be re-executed or if the current node should be stopped. However, there are cases where the user knows that a child node doesn't need to be re-executed after the first time. To address this, memory nodes have been introduced to the Behavior Tree, which store the success or failure of a child node and prevent it from being re-executed until the Sequence or Fallback finishes executing. These memory nodes are represented in the Behavior Tree by a symbol "!" added to the node's graphic representation. 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.

Similarly, we can identify other nodes such as OnNodeValueEqual and OnNodeValueChange, in the context of the OnNodeValueEqual node, which is employed 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 if so, or **a failure status** otherwise. This node is widely utilized, as exemplified by its prevalence in figures X, Y, and Z, owing to its ability to verify if



a node has returned the expected value. Specifically, it is commonly used to verify that a given OPC UA node has returned a Boolean variable with a value of "true."

3.5 Mock environment results

To ensure that the implementation is capable of handling the workflow objectives, comprehensive testing and simulation will be conducted on the developed implementations. The complete POLYLINE project will also be integrated and tested within the BMW production structures to verify its functionality and performance within a real-world production environment, however, testing within BMW's facilities necessitates the collaboration of numerous companies, and it was forecasted that it may not be feasible to complete the testing prior to the submission of this research. Hence, simulation has been deemed as the primary testing tool to evaluate this research viability and potential for implementation.

In this research, the software development methodology adopted is Test-Driven Development through automated mock testing using the GoogleTest (gtest)³ frameworks in C++. This approach has become widely popular in the software engineering community, due to its demonstrated efficacy and effectiveness over time. TDD 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++. They can be used to automate mock testing, which is a technique used to simulate the behavior of dependent components in a system under test. This allows developers to test individual units of code in isolation, and to verify their behavior under various conditions, without having to rely on the actual implementation of the dependent components. In gtest, automated mock testing can be performed by using mock objects to simulate the behavior of dependent components in the system under test. By using these mock objects, developers can write tests that verify the behavior of their code in isolation, without having to worry about the implementation of the dependent components.

An experimental environment was established with the purpose of assessing the performance of behavior trees in an industrial context. The evaluation involved conducting a suite of exercises that were designed to simulate real-world scenarios and elicit both successful and unsuccessful outcomes. These exercises were carefully

³ <http://google.github.io/googletest/>

crafted to challenge the behavior trees and test their ability to respond appropriately in different situations. The results of these exercises were used to determine the overall functionality of the behavior trees and to identify any areas for improvement. The ultimate goal of this evaluation was to ensure that the behavior trees were fully operational and capable of meeting the requirements of the industrial setting.

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 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 (Atanasijevic, 2020), 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 through the blackboard between the nodes and the real machines in the system, allowing for testing without the need for actual physical machines. The mock objects also simulate the signals received through the blackboard, which serves as a shared memory space for storing signals from the machines. This allows for testing of the system's ability to process and store signals correctly.

This approach to testing provides several benefits. It allows for faster and more efficient testing, as well as a controlled environment for testing, as the mock objects can be programmed to simulate specific scenarios and conditions. Additionally, it makes it easier to identify and isolate problems in the system, as the mock objects can be programmed to simulate specific failure conditions. Overall, creating mock objects of the nodes is an effective and efficient way to test the functionality and reliability of a system.

In conclusion, GoogleTest (gtest) provide efficient and adaptable options for automating mock testing. The utilization of TDD through these frameworks serves as a robust basis for this research, with the thorough testing and simulation ensuring the validity and reliability of the developed implementations and the successful integration of the POLYLINE project within BMW's production structures.

3.6 Analytics - On site results

Although the project is still in its testing phase, this section aims to elucidate the preliminary discoveries and findings made during the initial testing stages of the project. Furthermore, it serves to verify the theories formulated prior to the commencement of the project's development and highlight areas that require further improvement.

A significant discovery was the recognition of the potential offered by behavior trees in this project, particularly with regards to their modular and easily interpretable nature. This was observed due to the fact that the real-world environment in the factory was still undergoing development and not all machines could be tested, which resulted in multiple revisions to the final [tree.xml] file as various sequences of machines were implemented and individuals external to the research project became involved. This case study serves to illustrate the advantages of the modular structure of behavior trees in interdisciplinary environments, as they enhance human readability and manipulation of the system in general.

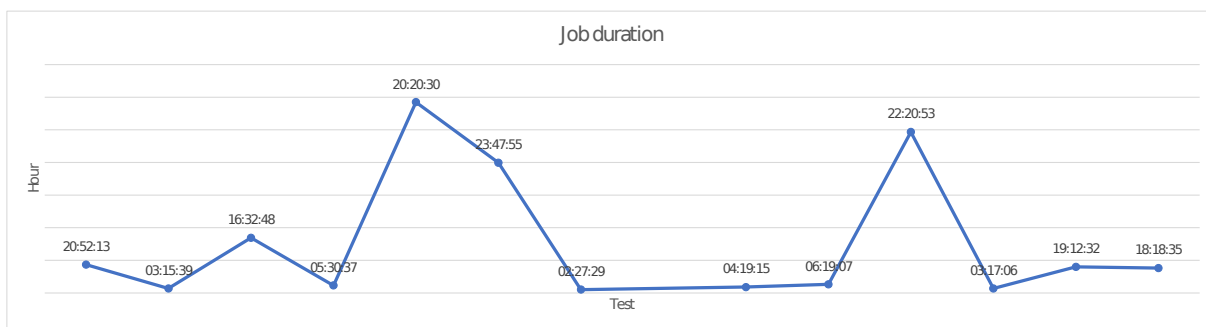
The study identified a key advantage of using Behavior Trees in this project, specifically in their modular and highly interpretable nature. This attribute proved to be especially useful as the real-world environment in the factory was still undergoing development and not all machines could be tested. As a result, multiple revisions were made to the final Behavior Tree file (tree.xml) to incorporate various sequences of machines and accommodate external contributors to the project. The hierarchical nature of Behavior Trees facilitated the isolation and resolution of issues, leading to more streamlined development and debugging of the system. This case study demonstrates the advantages of using Behavior Trees in designing complex systems that require adaptability and modularity, as they can be easily adjusted and fine-tuned to meet specific requirements and handle unexpected situations.

Although the workflow controller is still undergoing testing, preliminary data analysis has been possible by extracting information from the log files generated by the machines. However, generating aggregated analytics to enhance the speed and efficiency of data analysis has not yet been achieved. Nevertheless, a Python code was developed to extract meaningful insights from the log data and present them in a readable format. The data contained information about various events during the testing phase, including time stamps, errors or issues that occurred, and the overall behavior of the system. The log analysis provided valuable insights into the performance and behavior of the aggregator during the testing phase, predicting



the ultimate efficiency of the additive manufacturing system and identifying any bugs or issues that needed to be addressed.

Although the main testing was done through a simulation in a mock environment, recent on-site testing revealed additional insights into the project, verifying certain theoretical expectations. In this thesis, the findings from the on-site testing phase will be discussed, offering valuable insights into the project's Key Performance Indicators and the potential benefits of automation in the additive manufacturing field. It should be noted that these findings are from the testing phase and may vary in the final outcome, but they still provide crucial information for informed decision-making and ensuring the final product meets user needs and expectations.



Word, Percentage
error, 9.92%
warning, 19.38%
FAILURE, 0.00%
SUCCESS, 0.00%

4 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 and enhancement of Behavior Trees as a model of computation for constructing logic-based models of production lines in additive manufacturing. While the Behavior Trees model used in this research has shown promising results, there is still room for further investigation and enhancement of its application. One possibility could be to explore the implementation of different libraries to perform behavior tree modeling, which may offer additional features or advantages over the current approach. This could provide valuable insights into the best ways to construct and optimize behavior trees for this specific use case.
2. Aggregation of automatic analytics to improve performance analysis of the additive manufacturing production line. As explained in previous sections, at present, the ability to generate aggregated analytics from the MES has not been achieved. The goal would be to provide real-time insights into key performance metrics, such as production rates, defect rates, and energy consumption, and to identify areas where performance can be improved. Such automatic analytics could also be used to detect and diagnose potential issues before they become critical, allowing for proactive maintenance and repairs. This would ultimately lead to more efficient, reliable, and cost-effective additive manufacturing processes.
3. Expansion of additional key performance indicators (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. Continued research on communication protocols implemented in IIoT, specifically in the context of additive manufacturing. This could involve further analysis and comparison of different protocols, as well as the development and testing of new protocols that may better suit the specific requirements of additive manufacturing.



5. Improve the deliverables of the project. Particularly with regard to on-site implementation of the controller at the factory there is still room for improvement when connecting to the plants net and when setting necessary preconditions. As part of the effort to address these challenges, it's anticipated that the graphical user interface (GUI) will be enhanced, 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 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 critical 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.

5 Conclusions

Additive manufacturing possesses the potential to revolutionize the manufacturing industry by providing a multitude of benefits like accelerated prototyping, exceptional adaptability, and heightened efficiency. 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. This thesis presents a cloud workflow controller based on the implementation Behavior Trees that addresses these challenges and represents a step forward in the field of automation in additive manufacturing. The controller offers a systematic and effective way to manage the complex processes involved in the designated workflow. The goal of this controller is to automate an additive manufacturing plant in the BMW installations in Munich.

The research in this thesis provides valuable insights into the potential of automation in additive manufacturing, including the use of cloud-based solutions. The thesis also includes a comprehensive study of communication protocols in the Industrial Internet of Things and a thorough data analysis to understand the initial performance of the implementation site. The workflow controller was 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 various stages of the process. Furthermore, the results indicated that the proposed cloud workflow controller has the potential to increase efficiency and reduce manual intervention in the Additive Manufacturing process, contributing to the advancement of Industry 4.0.

The conclusions in this study show that Behavior Trees are a great possibility to have into account as they have shown that they can support the complex workload and decision making derived fr adapt to rapidly changing situations within the manufacturing environment, enabling operators to modify decision-making trees on the fly without requiring significant time or resources. This makes it easier to integrate new machinery into the system without undergoing significant modifications.

The research conducted in this study demonstrates that behavior trees, when combined with blackboard architecture, are an effective implementation for enabling communication in manufacturing systems. The flexibility of behavior trees to dynamically alter decision-making logic during runtime and incorporate new equipment into the system without significant overhead costs is particularly valuable. Additionally, the blackboard architecture provides a flexible framework for integrating diverse systems, enabling distributed problem-solving within a



complex system by allowing independent knowledge sources to access and modify a common knowledge store.

Overall, this research represents an important step towards optimizing the implementation of Additive Manufacturing technology and highlights the need for further research and development in this field. this research represents a significant advancement in the understanding and application of automation in additive manufacturing, providing valuable insights and advancements in the field.

In conclusion, this master thesis's findings demonstrate the potential for improving the efficiency and automation of the Additive Manufacturing process through automation, particularly with the implementation of a cloud workflow controller utilizing Industrial Internet of Things solutions. The research explored various models of computation, with a focus on the advantages of using Behavior Trees to organize and manage the complex processes involved in the production line. The study also conducted a comprehensive analysis of communication protocols to evaluate the suitability of the different protocols for the intended workflow. The results of this study provided valuable insights into the most suitable communication protocols for future research in this field.

6 Bibliography

1. (3YOURMIND, 2023)
3YOURMIND. (2023). 3YOURMIND. 3yourmind.com. Retrieved January 17, 2023, from <https://www.3yourmind.com/production-and-quality-management>
2. (Abdellatif et al., 2020)
Abdellatif, M., Tighilt, R., Belkhir, A., Moha, N., Guéhéneuc, Y.-G., & Beaudry, É. (2020). A multi-dimensional study on the state of the practice of REST APIs usage in Android apps. *Automated Software Engineering*, 27(3–4), 187–228. <https://doi.org/10.1007/s10515-020-00272-9>
3. (Agis et al., 2020)
Agis, R. A., Gottifredi, S., & García, A. J. (2020). An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, 155(113457), 113457. <https://doi.org/10.1016/j.eswa.2020.113457>
4. (Almada-Lobo, 2016)
Almada-Lobo, F. (2016). The Industry 4.0 revolution and the future of Manufacturing Execution Systems (MES). *Journal of Innovation Management*, 3(4), 16–21. https://doi.org/10.24840/2183-0606_003.004_0003
5. (Azzedin & Alhazmi, 2023)
Azzedin, F., & Alhazmi, T. (2023). Secure data distribution architecture in IoT using MQTT. *Applied Sciences (Basel, Switzerland)*, 13(4), 2515. <https://doi.org/10.3390/app13042515>
6. (Bauer et al., 2021)
Bauer, H., Höppner, S., Iatrou, C., Charania, Z., Hartmann, S., Rehman, S.-U., Dixius, A., Ellguth, G., Walter, D., Uhlig, J., Neumärker, F., Berthel, M., Stolba, M., Kelber, F., Urbas, L., & Mayr, C. (2021). Hardware implementation of an OPC UA server for industrial field devices. *ArXiv [Cs.AR]*. <https://doi.org/10.48550/ARXIV.2105.00789>
7. (Bhatt et al., 2020)
Bhatt, P. M., Malhan, R. K., Shembekar, A. V., Yoon, Y. J., & Gupta, S. K. (2020). Expanding capabilities of additive manufacturing through use of robotics technologies: A survey. *Additive Manufacturing*, 31(100933), 100933. <https://doi.org/10.1016/j.addma.2019.100933>
8. (Butt, 2020)
Butt, J. (2020). Exploring the interrelationship between additive manufacturing and Industry 4.0. *Designs*, 4(2), 13. <https://doi.org/10.3390/designs4020013>
9. (Chris, 2019)
Chris. (2019, May 6). Efficient IIoT communications: A comparison of MQTT, OPC-UA, HTTP, and modbus. *Cirrus Link*. <https://cirrus-link.com/efficient-iiot-communications-a-comparison-of-mqtt-opc-ua-http-and-modbus/>
10. [2] (Champanand & Dunstan, 2019)
Champanand, A. J., & Dunstan, P. (2019). The behavior tree starter kit. In *Game AI Pro 360* (pp. 27–46). CRC Press.
11. [3] (Colledanchise, 2017)
Colledanchise, M. (2017). Behavior trees in robotics. *Diva-portal.org*. Retrieved January 9, 2023, from <https://www.diva-portal.org/smash/get/diva2:1078940/FULLTEXT01.pdf>
12. [4] (Colledanchise & Ögren, 2017)
Colledanchise, M., & Ögren, P. (2017). Behavior trees in Robotics and AI: An introduction. *ArXiv [Cs.RO]*. <https://doi.org/10.48550/ARXIV.1709.00084>
13. [5] (Cooper & Lemaignan, 2022)

- Cooper, S., & Lemaignan, S. (2022). Towards using behaviour trees for long-term social robot behaviour. 2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI), 737–741.
14. (Delli & Chang, 2018)
Delli, U., & Chang, S. (2018). Automated process monitoring in 3D printing using supervised machine learning. *Procedia Manufacturing*, 26, 865–870. <https://doi.org/10.1016/j.promfg.2018.07.111>
15. (Derhamy et al., 2017)
Derhamy, H., Ronnholm, J., Delsing, J., Eliasson, J., & van Deventer, J. (2017). Protocol interoperability of OPC UA in service oriented architectures. 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), 44–50.
16. [6] (Dijkstra, 1968)
Dijkstra, E. W. (1968). Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3), 147–148. <https://doi.org/10.1145/362929.362947>
17. (Dizdarević et al., 2019)
Dizdarević, J., Carpio, F., Jukan, A., & Masip-Bruin, X. (2019). A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration. *ACM Computing Surveys*, 51(6), 1–29. <https://doi.org/10.1145/3292674>
18. [7] (Dorigo et al., 2013)
Dorigo, M., Floreano, D., Gambardella, L. M., Mondada, F., Nolfi, S., Baaboura, T., Birattari, M., Bonani, M., Brambilla, M., Brutschy, A., Burnier, D., Campo, A., Christensen, A. L., Decugniere, A., Di Caro, G., Ducatelle, F., Ferrante, E., Forster, A., Gonzales, J. M., ... Vaussard, F. (2013). Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4), 60–71. <https://doi.org/10.1109/mra.2013.2252996>
19. (Drahos et al., 2018)
Drahos, P., Kucera, E., Haffner, O., & Klimo, I. (2018). Trends in industrial communication and OPC UA. 2018 Cybernetics & Informatics (K&I).
20. [8] (Dromey, 2001)
R.G.Dromey, (2001). Genetic Software Engineering – Simplifying Design Using Requirements Integration, IEEE Working Conference on Complex and Dynamic Systems Architecture, Brisbane.
21. (Eiger API V3, n.d.)
Eiger API V3. (n.d.). Eiger.Io. Retrieved February 4, 2023, from <https://www.eiger.io/developer>
22. (Faconti, 2022)
Davide Faconti. 2022. BehaviorTree.CPP library Documentation. <https://www.behaviortree.dev>
23. (Faconti, 2019)
Faconti, D. Groot. <https://github.com/BehaviorTree/Groot>. 2019.
24. (Fielding, 2000)
Roy Fielding. 2000. “Architectural Styles and the Design of Network-based Software Architectures,” 128. University of California, Irvine, PhD Dissertation. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation_2up.pdf
25. (Francesca et al., 2014)
Francesca G, Brambilla M, Brutschy A, Trianni V, Birattari M. 2014. AutoMoDe: a novel approach to the automatic design of control software for robot swarms. *Swarm Intelligence* 8(2):89–112 DOI 10.1007/s11721-014-0092-4.
26. (Freens et al., 2015)
Freens, J. P. N., Adan, I. J. B. F., Pogromsky, A. Y., & Ploegmakers, H. (2015). Automating the production planning of a 3D printing factory. 2015 Winter Simulation Conference (WSC), 2136–2147.
27. (Friihwirth et al., 2015)
Friihwirth, T., Pauker, F., Fernbach, A., Ayatollah, I., Kastner, W., & Kittl, B. (2015). Guarded state machines in OPC UA. IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society, 004187–004192.
28. (Gao et al., 2019)

- Gao, W., Zhang, Y., Ramanujan, D., Ramani, K., Chen, Y., Williams, C. B., Wang, C. C. L., Shin, Y. C., Zhang, S., & Zavattieri, P. D. (2019). The status, challenges, and future of additive manufacturing in engineering. *Computer Aided Design*, 69, 65–89. https://www.academia.edu/38520195/The_status_challenges_and_future_of_additive_manufacturing_in_engineering
29. (Ghzouli et al., 2020)
Ghzouli, R., Berger, T., Johnsen, E. B., Dragule, S., & Wasowski, A. (2020). Behavior Trees in action: A study of robotics applications. *ArXiv [Cs.RO]*. <https://doi.org/10.48550/ARXIV.2010.06256>
30. (Ghzouli et al., 2022)
Ghzouli, R., Dragule, S., Berger, T., Johnsen, E. B., & Wasowski, A. (2022). Behavior Trees and state machines in robotics applications. In *arXiv [cs.RO]*. <http://arxiv.org/abs/2208.04211>
31. (Heckel et al., 2010)
Heckel, F. W. P., Youngblood, G. M., & Ketkar, N. S. (2010). Representational complexity of reactive agents. *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*.
32. (Horstkotte et al., 2021)
Horstkotte, R., Bruning, B., Prümmer, M., Arntz, K., & Bergs, T. (2021). Determination of the level of automation for additive manufacturing process chains. *Hannover: publish-Ing*. <https://doi.org/10.15488/11257>
33. (Ibrahim et al., 2018)
Ibrahim, H., Jahadakbar, A., Dehghan, A., Moghaddam, N., Amerinatanzi, A., & Elahinia, M. (2018). In vitro corrosion assessment of additively manufactured porous NiTi structures for bone fixation applications. *Metals*, 8(3), 164. <https://doi.org/10.3390/met8030164>
34. (Isla, 2005)
Isla, D. (2005, March 11). GDC 2005 proceeding: Handling complexity in the Halo 2 AI. *Game Developer*. <https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai>
35. (Isla, 2008)
Isla, D (2008). "Building a Better Battle: Halo 3 AI Objectives". In: *Game Developers Conference*. 2008.
36. [16] (S. Jones et al., 2018)
Jones, S., Studley, M., Hauert, S., & Winfield, A. (2018). Evolving behaviour trees for swarm robotics. In *Distributed Autonomous Robotic Systems* (pp. 487–501). Springer International Publishing.
37. [17] (Jones, 2020)
Jones, S. W. (2020). Onboard evolution of human-understandable behaviour trees for robot swarms. *University of Bristol*. Retrieved January 9, 2023, from <https://research-information.bris.ac.uk/en/studentTheses/onboard-evolution-of-human-understandable-behaviour-trees-for-rob>
38. (Industrie 4.0 - BMBF, 2011)
Industrie 4.0 - BMBF. (2011). Bundesministerium für Bildung und Forschung - BMBF. Retrieved January 20, 2023, from <https://www.bmbf.de/bmbf/de/forschung/digitale-wirtschaft-und-gesellschaft/industrie-4-0/industrie-4-0>
39. (Kellett, 2012)
Kellett, P. (2012). Additive Manufacturing and Automation Will 3D Printing Displace Automation Technologies? *Association for Advancing Automation*. https://www.automate.org/userAssets/riaUploads/file/Additive_Manufacturing_and_Automation.pdf
40. (Košťál et al., 2019)
Košťál, Peter & Mudriková, Andrea & Michal, Dávid. (2019). Group technology in the flexible manufacturing system. *MATEC Web of Conferences*. 299. 02001. 10.1051/mateconf/201929902001.
41. (Kim et al., 2015)

- Kim, D. B., Witherell, P., Lipman, R., & Feng, S. C. (2015). Streamlining the additive manufacturing digital spectrum: A systems approach. *Additive Manufacturing*, 5, 20–30. <https://doi.org/10.1016/j.addma.2014.10.004>
42. [18] (Klockner, 2013)
Klockner, A. (2013). Behavior Trees for UAV Mission Management. Dlr.de. Retrieved January 9, 2023, from <https://elib.dlr.de/91679/1/kloeckner2013behavior.pdf>
43. [19] (Kuckling et al., 2021)
Kuckling, J., van Pelt, V., & Birattari, M. (2021). Automatic modular design of behavior trees for robot swarms with communication capabilities. In *Applications of Evolutionary Computation* (pp. 130–145). Springer International Publishing.
44. (Kumar, 2019)
Kumar, S. (2019, October 28). Supercharge your REST APIs with protobuf. The Startup. <https://medium.com/swlh/supercharge-your-rest-apis-with-protobuf-b38d3d7a28d3>
45. (Ladegourdie & Kua, 2022)
Ladegourdie, M., & Kua, J. (2022). Performance analysis of OPC UA for industrial interoperability towards Industry 4.0. *IoT*, 3(4), 507–525. <https://doi.org/10.3390/iot3040027>
46. (Legarda Herranz, 2021)
Legarda Herranz, G. (2021). Evolution of behaviour trees for collective transport with robot swarms. Universitat Politècnica de Catalunya.
47. (Ligot et al., 2020)
Ligot, A., Kuckling, J., Bozhinoski, D., & Birattari, M. (2020). Automatic modular design of robot swarms using behavior trees as a control architecture. *PeerJ. Computer Science*, 6(e314), e314. <https://doi.org/10.7717/peerj-cs.314>
48. (Marzinotto et al., 2014)
Marzinotto, A., Colledanchise, M., Smith, C., & Ogren, P. (2014). Towards a unified behavior trees framework for robot control. 2014 IEEE International Conference on Robotics and Automation (ICRA), 5420–5427.
49. (Liu et al., 2018)
Liu, Y., Wang, L., & Vincent Wang, X. (2018). Cloud manufacturing: latest advancements and future trends. *Procedia Manufacturing*, 25, 62–73. <https://doi.org/10.1016/j.promfg.2018.06.058>
50. (Maia et al., 2020)
Maia, R. F., Bálsamo, Â. J., Lopes, G. A. W., Massote, A. A., & Lima, F. (2020). Evaluation of OPC-UA communication in an autonomous advanced manufacturing cell implementation. *Gestão & Produção*, 27(4). <https://doi.org/10.1590/0104-530x5414-20>
51. (Mies et al., 2016)
Mies, D., Marsden, W., & Warde, S. (2016). Overview of additive manufacturing informatics: “A digital thread.” *Integrating Materials and Manufacturing Innovation*, 5(1), 114–142. <https://doi.org/10.1186/s40192-016-0050-7>
52. (Nematollahi et al., 2019)
Nematollahi, M., Toker, G., Saghaian, S. E., Salazar, J., Mahtabi, M., Benafan, O., Karaca, H., & Elahinia, M. (2019). Additive manufacturing of Ni-rich NiTiHf20: Manufacturability, composition, density, and transformation behavior. *Shape Memory and Superelasticity*, 5(1), 113–124. <https://doi.org/10.1007/s40830-019-00214-9>
53. (Atanasijevic, 2020)
Atanasijevic, Tatjana. (2020). HOW TO CREATE A SUCCESSFUL USER EXPERIENCE (UX) ON THE HAPPY PATH AND THE UNHAPPY PATH. 10.13140/RG.2.2.10387.71207.
54. (Tadewos et al., 2019)
Tadewos, T. G., Shamgah, L., & Karimoddini, A. (2019). Automatic safe behaviour tree synthesis for autonomous agents. 2019 IEEE 58th Conference on Decision and Control (CDC), 2776–2781.
55. (Thomas & Gilbert, 2014)

- Thomas, D. S., & Gilbert, S. W. (2014). Costs and cost effectiveness of additive manufacturing. National Institute of Standards and Technology.
56. (Petch, 2020)
Petch, M. (2020, April 29). 3D Printing Community responds to COVID-19 and Coronavirus resources. 3D Printing Industry. <https://3dprintingindustry.com/news/3d-printing-community-responds-to-covid-19-and-coronavirus-resources-169>
57. (Petrovic et al., 2021)
Petrovic, Nenad & Radenković, Maša & Cvetkovic, Stevica & Rancic, Dejan. (2021). Model-driven automated gMock test generation for automotive software industry.
58. (Rao et al., 2020)
Rao, M., Lynch, L., Coady, J., Toal, D., & Newe, T. (2020). Integration of an MES and AIV using a LabVIEW middleware scheduler suitable for use in Industry 4.0 applications. Applied Sciences (Basel, Switzerland), 10(20), 7054. <https://doi.org/10.3390/app10207054>
59. (Rodríguez et al., 2016)
Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. C., Canali, L., & Percannella, G. (2016). REST APIs: A large-scale analysis of compliance with principles and best practices. In Lecture Notes in Computer Science (pp. 21–39). Springer International Publishing.
60. (Schiekofer et al., 2018)
Schiekofer, R., Scholz, A., & Weyrich, M. (2018). REST based OPC UA for the IIoT. 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 1, 274–281.
61. (Seiffert et al., 2022)
Seiffert, L., Szodrok, J., Ghofrani, J., & Wiczorek, K. (2022). Remote technologies as common practice in industrial maintenance: What do experts say? Telecom, 3(4), 548–563. <https://doi.org/10.3390/telecom3040031>
62. (Shah, 2022)
Shah, A. (2022). Emerging trends in robotic aided additive manufacturing. Materials Today: Proceedings, 62, 7231–7237. <https://doi.org/10.1016/j.matpr.2022.03.680>
63. (Sher, 2020)
Sher, D. (2020, June 25). BMW Additive Manufacturing Campus consolidates skills at a single site. 3D Printing Media Network - The Pulse of the AM Industry; 3D Printing Media Network. <https://www.3dprintingmedia.network/bmw-additive-manufacturing-campus-consolidates-skills-at-a-single-site/>
64. (STEP 7 professional (TIA portal), 2023)
STEP 7 professional (TIA portal). (2023). <https://mall.industry.siemens.com/mall/es/es/Catalog/Products/10314843>
65. (Stock & Seliger, 2016)
Stock, T., & Seliger, G. (2016). Opportunities of sustainable manufacturing in industry 4.0. Procedia CIRP, 40, 536–541. <https://doi.org/10.1016/j.procir.2016.01.129>
66. (Wang et al., 2019)
Wang, Y., Lin, Y., Zhong, R. Y., & Xu, X. (2019). IoT-enabled cloud-based additive manufacturing platform to support rapid product development. International Journal of Production Research, 57(12), 3975–3991. <https://doi.org/10.1080/00207543.2018.1516905>
67. (Wolde & Boltana, 2021)
Wolde, B. G., & Boltana, A. S. (2021). REST API composition for effective testing the cloud. Journal of Applied Research and Technology, 19(6), 676–693. <https://doi.org/10.22201/icat.24486736e.2021.19.6.924>
68. (Zhang et al., 2021)
Zhang, W., Deng, X., & Cai, J. (2021). Intelligent automatic 3D printing machine based on wireless network communication. Wireless Communications and Mobile Computing, 2021, 1–14. <https://doi.org/10.1155/2021/1778>

