# Implementing Automation Services for a Process Plant via OPC UA

## DIPLOMARBEIT

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Diplom-Ingenieurs (Dipl.-Ing.)

unter der Leitung von

Univ.-Prof. Dr.sc.techn. Georg Schitter
Dipl.-Ing. Martin Melik-Merkumians

eingereicht an der

Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
Institut für Automatisierungs- und Regelungstechnik

von

Yegor Chebotarev, BSc.
Matrikelnummer: 1329200

Wien, im März 2024

**Advanced Mechatronic Systems Group**
Gußhausstrasse 27-29, A-1040 Wien, Internet: http://www.acin.tuwien.ac.at

# Abstract

In the modern world, technologies change at an incredible speed, leading to a wide variety of products and their short lifetimes. Nowadays, industries need to adapt to this situation and possess the capability to produce various products with different batch sizes. The best approach to achieve this is through flexible Smart Factories that can be fast reconfigured according to new requirements. This has led to the search for new, modern approaches in the industry. The Plug and Produce strategy, based on the skill-based approach, is an innovative, promising, and challenging solution that can help achieve the required level of flexibility and redefine future standards in factory design.

This thesis attempts to take a step forward in this direction. It provides a brief overview of the main industrial trend - Industry 4.0. The theoretical part of the thesis represents the state of the art and provides a description of a concept. The practical part contains the implemented concept based on collected information and an evaluation of results.

The designed and implemented concept was tested on an existing laboratory setup. The setup consists of a redundant pipe system, which contains a large number of valves and two pumps. This pipe system connects several tanks, each equipped with its own set of sensors and additional equipment, such as heaters and mixers.

# Zusammenfassung

In der modernen Welt ändern sich Technologien mit unglaublicher Geschwindigkeit, was zu einer breiten Palette von Produkten und ihrer kurzen Lebensdauer führt. Heutzutage müssen sich Industrien dieser Situation anpassen und die Fähigkeit besitzen, verschiedene Produkte in unterschiedlichen Chargengrößen herzustellen. Der beste Ansatz, um dies zu erreichen, sind flexible Smart Factories, die schnell gemäß neuen Anforderungen neu konfiguriert werden können. Dies hat zur Suche nach neuen, modernen Ansätzen in der Industrie geführt. Die Plug-and-Produce-Strategie, basierend auf dem kompetenzbasierten Ansatz, ist eine innovative, vielversprechende und herausfordernde Lösung, die dazu beitragen kann, das erforderliche Maß an Flexibilität zu erreichen und zukünftige Standards in der Fabrikgestaltung neu zu definieren.

Diese Arbeit versucht, in diese Richtung einen Schritt vorwärts zu machen. Sie bietet einen kurzen Überblick über den Haupttrend der Industrie - Industrie 4.0. Der theoretische Teil der Arbeit stellt den Stand der Technik dar und liefert eine Beschreibung eines Konzepts. Der praktische Teil enthält das umgesetzte Konzept auf Grundlage gesammelter Informationen und eine Auswertung der Ergebnisse.

Das entworfene und umgesetzte Konzept wurde auf einem bestehenden Laboraufbau getestet. Der Aufbau besteht aus einem redundanten Rohrsystem, das eine große Anzahl von Ventilen und zwei Pumpen enthält. Dieses Rohrsystem verbindet mehrere Tanks, von denen jeder mit seinem eigenen Satz von Sensoren und zusätzlicher Ausrüstung wie Heizungen und Mischern ausgestattet ist.

# Contents

# Acronyms

| | |
|---|---|
| **AI** | Artificial intelligence. |
| **ANSI** | American National Standards Institute. |
| **AutomationML** | Automation Markup Language. |
| **CoAP** | Constrained Application Protocol. |
| **CPS** | Cyber-Physical System. |
| **EAS** | Evolvable Assembly Systems. |
| **ECC** | Execution Control Chart. |
| **ERP** | Enterprise Resource Planning. |
| **FB** | Function Block. |
| **GUI** | Graphical User Interface. |
| **HTTP** | HyperText Transfer Protocol. |
| **IDEAS** | Instantly Deployable Evolvable Assembly System. |
| **IEC** | International Electrotechnical Commission. |
| **IIoT** | Industrial Internet of Things. |
| **IoT** | Internet of Things. |
| **ISA** | International Society of Automation. |

| | |
|---|---|
| **IT** | Information Technology. |
| **MQTT** | Message Queuing Telemetry Transport. |
| **MRA** | The Machine Resource Agent. |
| **OPC UA** | Open Platform Communications Unified Architecture. |
| **ORA** | Ontologies for Robotics and Automation. |
| **PFC** | Procedure Function Chart. |
| **PLC** | Programmable Logic Controllers. |
| **SCADA** | Supervisory Control and Data Acquisition. |
| **SEM** | Skill-based Engineering Model. |
| **SOA** | Service Oriented Architecture. |
| **UML** | Unified Modeling Language. |
| **URI** | Uniform Resource Identifier. |

# List of Figures

# List of Tables

# CHAPTER 1

# Introduction

This chapter provides an understanding of Industry 4.0 and the concept of Smart Factories, emphasizing their impact on manufacturing through advanced automation, digital technologies, and real-time production flexibility for small batches and individual items.

## 1.1 Industry 4.0

With the evolution of human society humanity passed through four evident and clearly observable industrial revolutions:

*The First Industrial Revolution* - which occurred in the second half of the 18th century. The key feature of this industrial revolution was based on the introduction of mechanical equipment driven by water and steam-powered machines, and the introduction of more optimized forms of working, e.g., mechanized cotton spinning (power loom), and stationary steam engine [1–3].

*The Second Industrial Revolution* - in the early part of the 20th century, was characterized by the introduction of steel and the use of electricity in factories. This revolution based on new concepts of mass production such as the first conveyor belt was introduced as a way of increasing productive output [1–3].

*The Third Industrial Revolution* - also called digital, slowly began in the second half of the 20th century along with the development of technology and based on the use of electronics and digital technology in factories to further automate and increase the capacity of manufacturing [1–3].

*The Fourth Industrial Revolution*, or Industry 4.0 - that has come about over the last few decades, promotes a trend of automation through the use of digital technology and new levels of interconnectivity with the aid of the Internet of Things

(IoT) and cloud computing, allowing real-time access to data and the introduction of cyber-physical systems [1, 3].

A vision of "Industry 4.0" was first time announced and presented to a wide audience at the Hanover Fair in the spring of 2011 by senior IT scholars and political representatives. It was announced as a far-reaching industrial paradigm shift for Germany based on new digital technologies [4]. The primary characteristics of Industry 4.0 include the establishment of significantly automated sectors by facilitating interaction between humans and machines. The adoption of Industry 4.0 is poised to enhance operational efficiency in terms of time, cost, and productivity for industries [5].

Technologically, the concept of Industry 4.0 is based on two main elements. A very flexible and largely autonomous network between physical systems and components of various types (production and products) with distributed and intelligent software systems and global data networks. The vision is aimed at the intelligent and purposeful use of data obtained during production and sales, using sensors and advanced assessment methods [4, 6].

Emerging concepts, such as digitalization, Internet of Things (IoT), and Cyber-Physical System (CPS), have gained significance across various industries, including manufacturing. These terms play a defining role in the Fourth Industrial Revolution [5]:

- *Enterprise Resource Planning (ERP)*: ERP tools facilitate the management of organizational information and business processes.
- *IoT*: IoT represents the connectivity between physical objects, such as sensors and machines, and the Internet.
- *IIoT*: IIoT encompasses the interconnections between people, data, and machines in the context of manufacturing.
- *Big data*: Big data refers to extensive sets of structured or unstructured data that can be collected, stored, organized, and analyzed to uncover patterns, trends, associations, and opportunities.
- *Artificial intelligence (AI)*: AI pertains to a computer's ability to perform tasks and make decisions that would traditionally require human intelligence.
- *M2M*: M2M stands for machine-to-machine and denotes communication between two separate machines through wireless or wired networks.
- *Digitization*: Digitization involves the process of gathering and converting various types of information into a digital format.

- *Smart factory*: A smart factory strategically employs Industry 4.0 technology, solutions, and methodologies.
- *Machine learning*: Machine learning signifies the capacity of computers to autonomously learn and enhance their performance through artificial intelligence, without explicit programming.
- *Cloud computing*: Cloud computing involves the use of interconnected remote servers on the Internet for storing, managing, and processing information.
- *Real-time data processing*: Real-time data processing signifies the ability of computer systems and machines to continuously and automatically process data, providing real-time or near-time outputs and insights.
- *Cyber-physical systems*: Cyber-physical systems, sometimes referred to as cyber manufacturing, describe Industry 4.0-enabled manufacturing environments that offer real-time data collection, analysis, and transparency across all aspects of manufacturing operations [6, 7].

Industry 4.0 is driving a profound transformation in manufacturing by promoting highly automated industries and facilitating human-machine interaction. This transformation is enabled by the flexible and autonomous network between physical systems, intelligent software systems, and global data networks. With Industry 4.0 manufacturing is evolving into a smart, interconnected ecosystem where real-time data processing and analysis are essential, ultimately reshaping the way businesses operate and innovate.

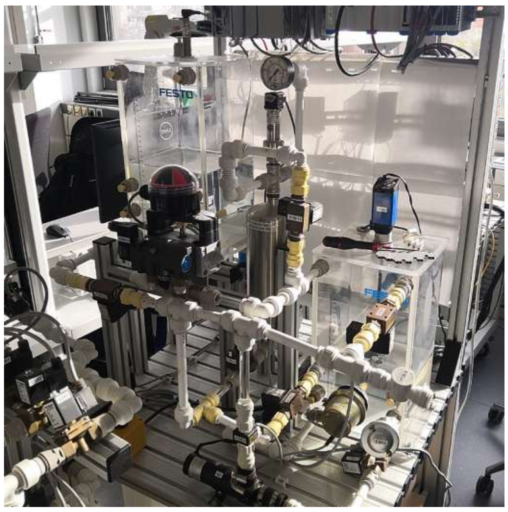
### 1.1.1 Smart Factory

With everyday increasing amount of customers and their expectations, there is a need for more flexible production. It can be achieved by developing more multipurpose factories. Dynamic reconfigurability and adaptability are crucial features of the future manufacturing systems [8]. In this way, a new level of automation should be achieved, by the introduction of methods of self-optimization, self-configuration, self-diagnosis, cognition, and intelligent support of workers in their increasingly complex work [9]. These terms are used in defining so-called concept of a Smart Factory. At some point, Smart Factory is a key point of Industry 4.0 because it should contain all other aspects that were described previously for successful operation. It is a symbiosis of network and software technologies in a way that all parts and components have intelligent communication with each other, and they are able to work and automatically coordinate with each other without or with only slight human influence.

Smart Factories are appealing due to their ability to handle small batches, even individual items. Industry 4.0 brings real-time, zero-setup-time production flexibility, meeting the need for personalization and mass customization in both consumer and business contexts. This means manufacturing thousands of identical parts and a single unique piece on the same line without extra costs or setup time [10].

## 1.2 Scope of the thesis

The modern market of technologies evolves with incredible speed. Together with a short lifecycle of goods, it leads to the production of fast-changing in-size batches of different products. The classical factory is designed for the long-term production of one type of product on an assembly line, and that is why incapable of a fast adaptation of a process. An upgrade or a reconfiguration of a plant takes a lot of time and oftentimes impossible without stopping for the entire service period. All these requirements demand to be flexible and have a possibility for a fast reconfiguration without the necessity to stop the facility which causes dead time and money losses [11].


(a) Two tanks laboratory setup.

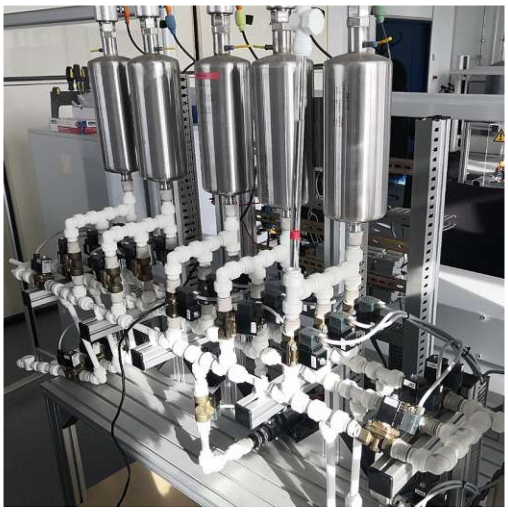

(b) Five tanks laboratory setup.

Figure 1.1: The left setup consists of several valves, a pump, and two tanks. The first tank (left tank) has a level sensor. The second (right tank) has a mixer, a heater, and a temperature sensor. The right system has a redundant pipe system, one pump, several valves, and five tanks. Each tank has a level sensor.

This work is focused on developing a flexible system for a batch process based on the current trends in the industry. The system should offer capabilities for fast reconfiguration, be fault-tolerant, and provide a user interface. The user interface should contain manual control, real-time visualization of a process, and data storage. This work is inspired by the work of M. Baierling [12], yet it uses alternative methodologies and solutions to address the research objectives.

This thesis is structured in the following manner. The second chapter provides an overview and analysis of modern concepts such as the Skill-based approach and Service Oriented Architecture. It also gives an idea of basic concepts such as batch processes, search algorithms, and graph analysis. The third chapter provides a theoretical description of a concept for the batch process that should be implemented and applied in a laboratory setup shown in Figure 1.1. It describes the problem statement, possible problems during implementation, and ways to solve them. The fourth chapter is a description of development tools, such as software, and also describes in detail the principle of operation of each key part of the project. The fifth chapter provides an analysis based on the results obtained after running the developed concept on the provided laboratory setup. Finally, the sixth chapter contains conclusions, analyzes the pros and cons of the concept, and gives an outlook for future work.

# CHAPTER 2

# State of the Art

This chapter gives an analysis of the related to the thesis topic paradigms, concepts, industrial standards, and possible solutions for the further implementation of a concept based on a flexibility and batch production approach. First, Service Oriented Architecture (SOA) is presented as an initial point for a flexible system. After that, the skill-based approach shows an effort to adapt the SOA paradigm for an industry application domain. A subchapter with various searching algorithms has the aim to solve a shortest path problem that can occur in a redundant pipe system. In the last part of the chapter, an overview of batch processes as well as batch recipes is given by virtue of relevant industrial standards.

## 2.1 Service Oriented Architecture

Service Oriented Architecture (SOA) holds various definitions, each complementary and tailored to specific audiences. When addressing IT architects, it is perceived as an architectural solution promoting loose coupling and reusability for integrating diverse systems. For developers, SOA is a programming paradigm where web services and contracts take center stage in designing for interoperability. The diversity in definitions arises from the need to convey SOA concepts to different stakeholders, recognizing the distinct perspectives of CEOs and programmers [13]. All of them have one common point, SOA is a paradigm for improved flexibility [14]. SOA refers to a model where automation logic is decomposed into discrete units, each constituting a distinct piece of business automation. These units, collectively forming a larger automation logic, can be distributed individually [15]. SOA is not a concrete architecture and not a specific tool or framework. It is a guiding principle that shapes concrete software architecture decisions [14]. The technical concept of SOA consists of:

– Services

– Interoperability

– Loose coupling

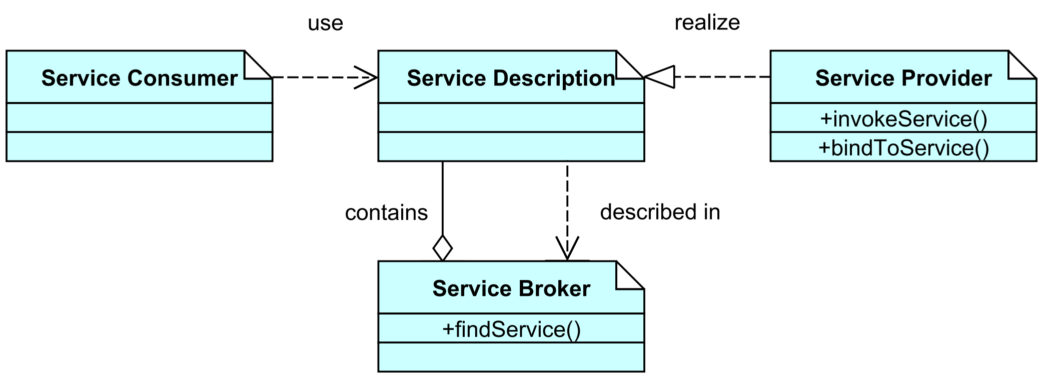


Figure 2.1: A meta-model of a SOA architectural style [16].

Figure 2.1 shows a Unified Modeling Language (UML) diagram of the meta-model of a SOA architectural style. This concept is grounded in an architectural style that outlines an interaction model involving three key participants: the *service provider*, responsible for publishing a service description and offering the service implementation; a *service consumer*, who can either directly use the Uniform Resource Identifier (URI) for the service description or locate the service description in a service registry to bind and invoke the service; and the *service broker*, which provides and manages the service registry, although public registries are currently less popular [16].

## 2.1.1 Services

SOA promotes the autonomy of individual units of logic without isolating them from each other. These units of logic are encouraged to adhere to principles that enable independent evolution while maintaining a necessary level of commonality and standardization. In SOA, these units are called *services* [15]. Basically, service is a fundamental element of SOA, representing a self-contained and logical business function in an IT. In technical terms, a service functions as an interface for one or more messages, which provide information and/or change the state of a related entity [13, 14]. The following list gives an overview of the main attributes:

- Self-Contained (independent, autonomous, autarkic). The service operates independently, encapsulating its functionality without dependence on other services. Despite this, certain dependencies are inevitable, e.g. services might share basic data types like strings [14].

- Visible/Discoverable. The service is publicly exposed and can be discovered by potential users, ensuring it is known and accessible. Often does exist a public place where the service and all detailed descriptions about it can be found [13, 14].

- Idempotent. The service's operations yield consistent results, maintaining predictability regardless of the number of invocations. Whether the service is called once or multiple times, the result remains unchanged. In essence, idempotence ensures that the outcome of a successfully executed request remains unaffected by the number of times it is carried out [14, 17].

- Interoperable. The service can smoothly communicate with other services and applications, ensuring compatibility. Interoperability is a core requirement of SOA and refers to the capacity of two or more systems or components to share information and effectively utilize the exchanged information [14, 18].

- Coarse-Grained. The service executes significant tasks, minimizing the need for frequent interactions. It is generally more advantageous to have a single service call handle the transfer of all required data between a provider and its consumer(s), rather than having multiple service calls processing the same volume of data [14].

- Reusable. The service's functionality is versatile, and suitable for use in different contexts or by multiple applications. Which means that each functionality should have a single implementation. SOA contributes to improved reusability by enabling various systems requiring a specific functionality to invoke the same service [14, 15].

The focus of a service can range from small to large, influencing the size and scope of the encapsulated logic. Additionally, service logic may include the logic offered by other services, leading to the composition of one or more services into a collective whole. As shown in Figure 2.2, when constructing an automation solution with services, each service has the capacity to encapsulate a task performed by an individual step or a sub-process comprising multiple steps. In certain scenarios, a service can encapsulate the entire process logic, and in such cases, the broader scope of these services may include the logic encapsulated by other services [15].

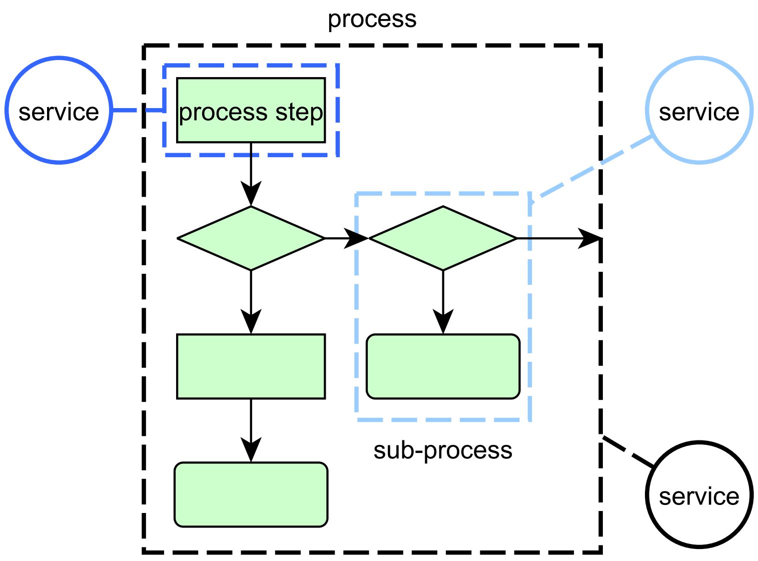


Figure 2.2: Configuration and a visual example of Services. A task addressed to a service can be small or large. That is why the size and scope of the represented logic of the service can vary. A service logic can contain the logic provided by other services. In this situation, a composition of one or more services is called a collective [15].

### 2.1.2 Loose coupling

The manufacturing industry uses an extensive array of interconnected and diverse systems, comprising numerous components that range from individual sensors to entire control or monitoring subsystems, varying widely in complexity. An agile automation system is expected to be a set of loosely coupled yet tightly integrated components, aligning with the desired characteristics of a distributed SOA system [19, 20].

SOA is designed for large distributed systems, where scalability and fault tolerance are critical for ensuring the maintainability of such systems. The concept of loose coupling is commonly utilized to address the demands of scalability, flexibility, and fault tolerance in systems. Systems need to manage errors and issues, and when they arise, it's crucial to minimize their impact. The essence of achieving these objectives lies in loose coupling, which involves minimizing dependencies. With reduced dependencies, modifications have limited effects, and the system can continue functioning even when certain parts are compromised. Minimizing dependencies enhances fault tolerance and flexibility [14].

*Asynchronous communication* is the most well-known example of achieving loose coupling in SOA. It allows services to communicate without requiring both the sender and receiver to be actively engaged at the same time. Asynchronous communication decouples services in time, as the sender and receiver do not need to be synchronized, which minimizes dependencies on the timing of service invocations. Also, services can send requests and continue their operations without waiting for an immediate response. This asynchronous nature allows each service to operate independently, improving flexibility. Another feature is allowing services to handle a large number of requests simultaneously without waiting for synchronous responses. If a service is temporarily unavailable, messages can be queued for later processing, preventing immediate disruptions [14].

*Heterogeneous Data Types.* The problem for the next example of loose coupling can be observed in a short example. A customer has a harmonized data type that must be extended with additional attributes. In a situation where this data type is shared with all systems, even if some of the systems are not interested in these extensions, the whole system must be updated to take into account these changes. With time this data type can become more complex and synchronization of the system becomes more expensive and time-consuming [14].

In the absence of harmonized data types, the introduction of data type mappings becomes necessary, encompassing both technical and semantic considerations. The usual approach involves the service provider specifying the data types used by its services, which service consumers are required to accept. It's crucial for consumers to avoid directly using the provider's data types in their source code. Instead, a recommended practice is for consumers to implement a lightweight mapping layer,

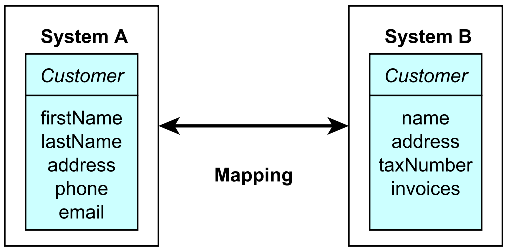


Figure 2.3: Decoupling by using different data types. Large distributed systems have no harmonized data types. As a result, different systems have different data types. Data type mapping helps to solve this problem. Heterogeneous data types can be accommodated through standardized data exchange formats such as XML or JSON. Services can understand and process data in various formats, enabling them to communicate despite differences in data types. Picture adapted from [14].

facilitating the translation of provider-specific data types to their own. However, this approach encounters scalability issues and contributes to heightened complexity. Heterogeneous data types imply that services can use different data formats or structures to communicate. This flexibility allows services to evolve independently, as changes in one service's data structure do not necessarily affect others [14].

A *mediator* is the last observed in this chapter example of loose coupling. It is a component that facilitates communication and coordination between services without them being directly aware of each other. It acts as an intermediary, promoting independence and reducing direct dependencies between services. This helps achieve a higher level of flexibility and adaptability in the system, as changes in one service are less likely to impact others due to the mediated communication [14].

There are two types of mediators. The first type provides the correct endpoint for a service call before it is sent. This type is often referred to as a broker or name server. The second type determines the right endpoint for a request after the consumer has sent it. In this scenario, the consumer sends the request to a symbolic name, and the infrastructure (network, middleware) guides the call to the appropriate system based on intelligent routing rules [14].

## 2.2 Agent, Task and Solution

The term *action* is used to describe an activity performed by an agent, encompassing actions like applying force, movement, perception, or communication to bring about a change in its environment and internal state. An agent, defined as any entity capable of interacting with its surroundings, acts purposefully when driven by a specific intended goal [21]. The environment of an agent may encompass other agents. The combination of an agent and its environment is referred to as a world [22].

For instance, an agent might be a fusion of a computational engine with physical sensors and actuators, commonly known as a robot, operating within a physical environment. It could involve pairing an advice-giving computer, an expert system, with a human who contributes perceptual information and performs tasks. An agent may also take the form of a program that functions in a purely computational environment, commonly called a software agent [22].

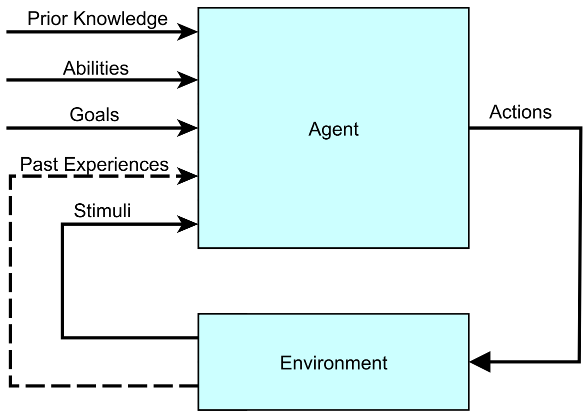


Figure 2.4: An *agent* interacting with an *environment*.The actions performed by an agent at any given moment are influenced by: *prior knowledge* about both the agent and the environment. *Stimuli* received from the current environment, which can include observations about the environment, as well as actions that the environment imposes on the agent. *Past experiences* derived from prior actions, stimuli, or other data that contribute to its learning. *Goals* that it aims to achieve or preferences regarding states of the world. *Abilities*, representing the primitive actions it can execute [22].

Usually, the problem to be solved or the task to be executed, along with the definition of a solution, is provided in an informal manner. The overall structure for solving problems using a computer is illustrated in Figure 2.5. To address a problem, the system designer needs to [22]:

– elaborate on the task and define what qualifies as a solution;

– represent the problem in a computationally understandable language;

– use the computer to compute an output, whether it's an answer for a user or a series of actions for the environment;

– interpret the output data as a solution to the problem.

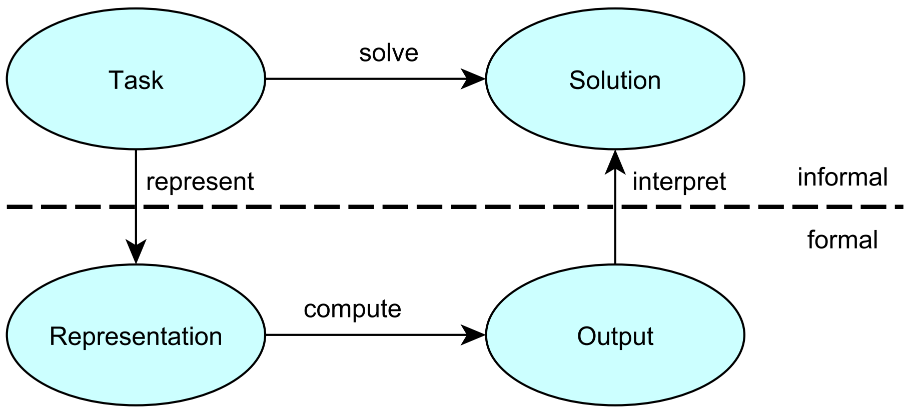


Figure 2.5: The role of representations in solving tasks. The image is from [22].

Given an informal task description, a designer should first establish the criteria for a *solution*. When dealing with well-defined tasks, the consideration shifts to the importance of accuracy and completeness in the provided answer. Often, the emphasis is not on obtaining the absolute best solution, but rather on achieving an adequate solution that satisfies certain criteria [22]. The four main classes of solutions are [22]:

*Optimal solution*: This is the best solution based on a specific measure of solution quality.

*Satisficing solution*: It is a solution that is deemed good enough according to predefined adequacy criteria. This is suitable for situations when an agent needs just some solution.

*Approximately optimal solution*: This solution comes close to the theoretically best quality measure, offering a practical alternative to optimal solutions.

*Probable solution*: While not guaranteed to be a solution, it is likely to be one, providing a precise approximation, especially in the context of satisficing solutions.

## 2.3 Skill-based Approach

Modern trends and an evolution of the industrial segment create conditions for reconsideration of traditional methodologies and approaches. This chapter discusses one of those new approaches that satisfy Industry 4.0 requirements.

### 2.3.1 Skill-based Engineering

The ongoing shift in manufacturing systems from mass production to mass customization, even reaching the extremes of batch size one production, demands more adaptable engineering solutions than those currently exist [23]. This adaptability often prompts discussions about flexibility and changeability. In this context, *flexibility* is defined as the system's capability to automatically adjust within a specified predefined range established during the system's design [11]. Presently, the majority of manufacturing systems have limited flexibility, primarily characterized by centralization, making it challenging to efficiently reconfigure an existing production line for the production of diverse product variants [23]. If a current manufacturing system's existing configuration cannot produce a new product, the addition or replacement of resources or software is necessary without replacing the entire system. If this process is simple and cost-effective, it indicates a high level of system *changeability* [11].

A new approach that describes relationships between Product, Process, and Resource via *Skill* is called Skill-based Engineering Model (SEM). Skills can be likened to services in SOA or methods in programming [24]. Skills are the process capabilities offered by equipment to carry out essential steps in the assembly process [25]. Designing systems based on the required skills for each step of the process is the primary goal of Skill-based Engineering. In this model, skill is a common point that satisfies the product's requirements and resource possibilities. Skill can represent either primitive functionality e.g., open/close valve, or less refined and more complex functionality like a pick-and-place process [11].

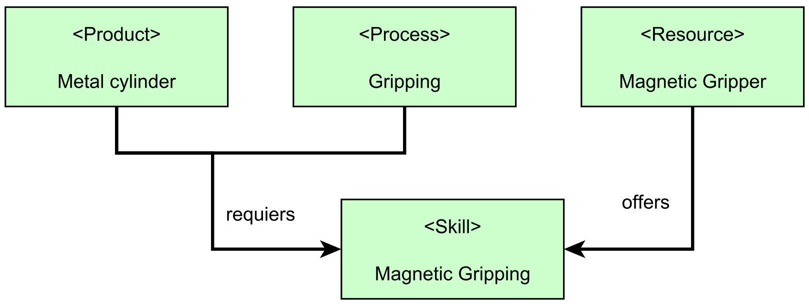


Figure 2.6: Key components of Skill-based Engineering. 1) The product to be manufactured, 2) The manufacturing process for this product, 3) The available resources capable of executing individual steps in this process, 4) The skill that connects these entities and is primarily associated with the desired process [11].

The way to avoid systems complexity and increase flexibility is to split complex (composite) skills into sequences of simple (atomic) skills [11, 26, 27]. *Atomic* *skills* represent fundamental skills at the most granular level (e.g. open/close valve), whereas *Composite skills* encompass a collection of elemental or lower-level composite skills (e.g. pick-and-place process). These composite skills implement a workflow-like sequence of processes that support sequential, parallel, and conditional execution of skill flows [27, 28]. However, SEM does not offer any recommendations about how raw skill descriptions should be. Furthermore, the skill's description represents a challenge because a detailed explanation of skills leads to increasing flexibility and whole system complexity and vice versa, a more common explanation makes the system less complex and less flexible. The way to solve this question is to find a balance between flexibility and complexity. Another problem that is linked with the skill's description is the matching problem. This problem contains tasks to satisfy process requirements with available skills [29].

The work [30] presents a modeling language for straightforward parameterized skills. A skill consists of a name (e.g., Move, Drill) and a set of parameters, which can include numerical values with defined minimum and maximum limits (e.g., work range from 0 to 30 mm), booleans, or enumerations (e.g., the enumeration of possible shapes to screw) [11, 29, 30].

The work also presents a basic algorithm to resolve the matching challenge when skills are defined using a specific language, where required and offered skills must share the same name, and each parameter of the required skill must be part of the set of parameters provided by the offered skill. However, this approach has a limitation as the matching problem can only be resolved for skills sharing identical names e.g., a Move skill can be matched with another skill named Move but not with one named LinearMove [29, 30].

To overcome this constraint, one strategy involves establishing relationships between skill names, such as explicitly stating that a LinearMove is a specific instance of a Move. Ontology-based approaches address this challenge, as seen in works like [11, 31, 32]. These approaches vary in their modeling techniques, whether through skills (representing the sequence of tasks for product manufacturing [31]) or by different approaches to product modeling (from an abstract symbolic representation to a complete geometric representation [32]) [29].

Authors of [11] proposes ontologies (Figure 2.7) as a promising method for defining and deriving composite skills from a collection of atomic skills. They also facilitate the definition of required skills for specific processes and the assurance of skills by particular resources. However, according to [11] ontologies have some limitations in their current form like limited standardized ontologies for industrial systems, computational complexity, and lack of calculation and simulation possibilities. In spite of these disadvantages, some ontologies like Ontologies for Robotics and Automation (ORA) [33] can be considered as reference [11].

Another solution for this problem is described in [26]. Because of increased

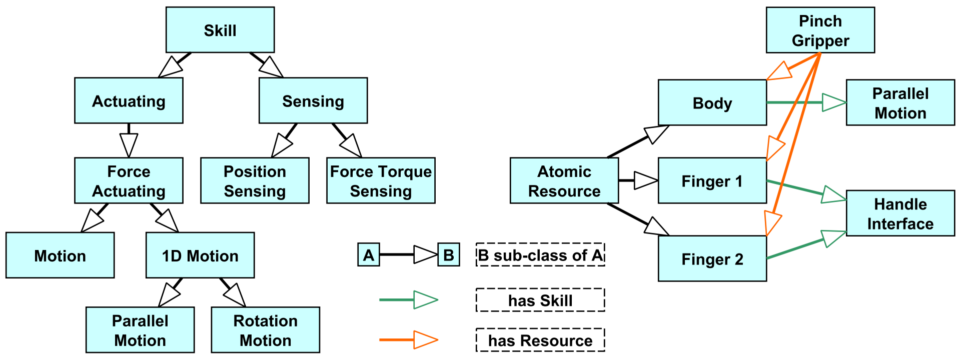


Figure 2.7: Examples of ontologies usage. The left image shows of skills definitions using OWL ontologies and illustrates their application to establish a skill taxonomy with various levels of abstraction, e.g. Actuating is a subtype of Skill, featuring a specialization named ForceActuating. The right image is an example of resources and skills definition and illustrates the definition of pinch gripper. It consists of three other resources, each providing specific atomic skills. Ontologies allow to infer this composite skill from the collection of atomic skills.

complexity in product workflows and skill requirement definitions, it was observed that certain requirements can not be aligned with any skill. This occurs due to the introduction of new requirements that the assembly system can not fulfill or when requirements are specified at the composite level [26, 27]. There are two potential solutions to this problem. The first is to integrate new skills into the system and the second is the use of patterns. Patterns present alternative representations of composite-level skill requirements, defining them as subsets of skill requirements (atomic or composite) that an assembly system could potentially fulfill, as illustrated in Figure 2.8 [26, 27].

IDEAS project is an example of the Skill-based approach implementation. The Instantly Deployable Evolvable Assembly System (IDEAS) project [25] aimed to offer an industrially feasible approach to Evolvable Assembly Systems (EAS). This evolvable system relies on principles of modularity, plug and produce, and agent (include skills) technology. The system is based on the concept of skills, which are implemented as function blocks following the IEC 61499 standard, ensuring a separation between data flow and process execution [26, 27]. Skills encapsulate assembly processes provided by mechatronic agents known as Resource Agents (or The Machine Resource Agent (MRA) [28]), tasked with overseeing the associated equipment as shown in Figure 2.8 [27]. These resource agents contain the atomic skills that can be performed by the mechatronic entity [28].

Another example that shows an implementation of skill skill-based engineering

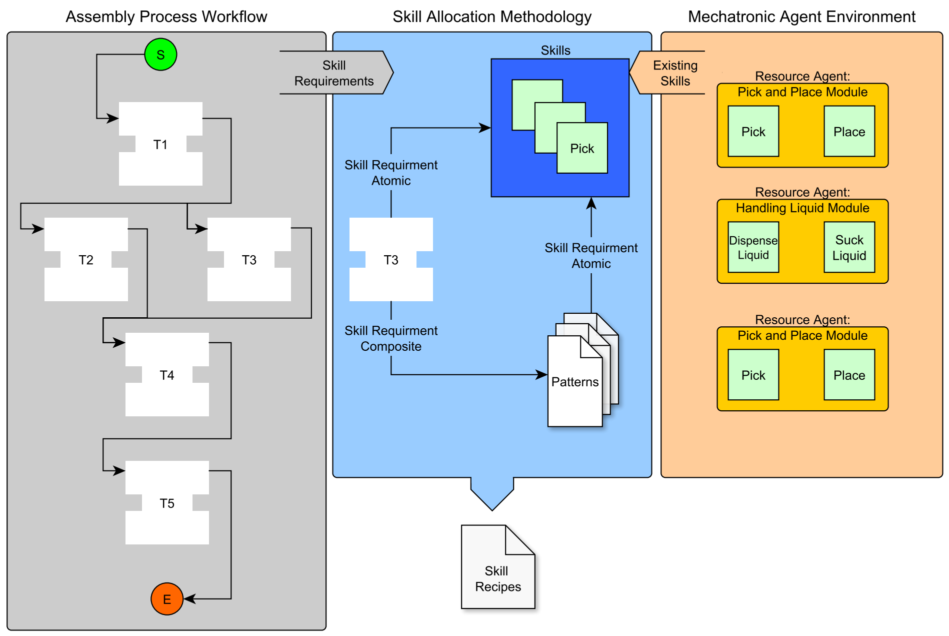

Assembly Process Workflow
Skill Allocation Methodology
Mechatronic Agent Environment
S
T1
T2
T3
T4
T5
E
Skill Requirements
Skills
Existing Skills
Pick
Skill Requirment Atomic
T3
Skill Requirment Atomic
Skill Requirment Composite
Patterns
Skill Recipes
Resource Agent: Pick and Place Module
Pick
Place
Resource Agent: Handling Liquid Module
Dispense Liquid
Suck Liquid
Resource Agent: Pick and Place Module
Pick
Place

Figure 2.8: Problem definition of skill allocation methodology with patterns [27].

model is presented in [23]. This work is focused on using of common interface that is independent of skill implementation and guarantees interoperability between various tools and programming standards. This gives a possibility of the construction of a system based on a hierarchical skill model, spanning from atomic to complex composite skills. The project involves the operation of four distinct devices from different vendors, communicating their skills through a common interface (OPC UA), e.g. control of one of the equipment elements is managed by a soft-PLC running CODESYS runtime, with its skill implemented in IEC 61131-3, while the remainder of the application is executed in IEC 61499 [23]. The utilization of an OPC UA interface as a shared skill description facilitates the abstraction of low-level functionality implementation, showcasing interoperability among diverse software implementations on various devices.

The works [34–36] also propose IEC 61499 as a standard for skills implementation. However, VDMA R+A OPC UA Demonstrator from [35] utilizes both standards IEC 61499 and IEC 61131-3. The EnAS [34] project is based on the SOA paradigm and has a goal to develop a flexible assembly line with a changeable layout, integrated with a mobile robot.

The definition and standardization of skills can vary between different systems and industries. Achieving a common understanding and agreement on skill definitions, especially at the composite level, may require industry-wide collaboration and has to be performed by the standardization organizations [11, 37, 38]. In summary, Skill-based Engineering provides a framework for designing agile and adaptable automation systems that can efficiently handle the complexity and variability present in modern manufacturing environments. It focuses on the skills required for specific tasks, promoting modularity, interoperability, and flexibility in automation system design.

### 2.3.2 Plug and Produce

The current situation in the industry is characterized by an increasing variety of products, decreasing batch sizes, and volatile demand. These conditions, coupled with the decreasing duration of product life cycles, resulting in a scenario where the need for adjustments and adaptations to manufacturing equipment becomes more frequent [39]. Traditionally factories are conventionally structured to accommodate large-scale production, with production lines frequently maintaining a consistent configuration throughout their operation [40]. Modern manufacturing systems commonly depend on a fixed or "hard-wired" centralized hierarchy of Programmable Logic Controllers (PLC). Consequently, to reconfigure the system, it is necessary to halt the plant, rewire the components, and reprogram the controllers [41].

First or one of the first mentions of the term *Plug and Produce* is introduced in [42]. Derived from the familiar *Plug and Play* concept in computing, *Plug and Produce* aims to simplify the commissioning process. This process involves the installation of new devices and the removal of devices without impacting others, managing the registration or deregistration of devices from the database, and eliminating the necessity for reprogramming [40, 42]. A proper description of mechatronic modules and their capabilities (skills) is essential for the *Plug and Produce* concept. Once implemented, these modules can directly offer their capabilities for utilization in production processes. Within this framework, skills are assessed against skill requirements and chosen to establish a new or modified system. Hence, they have a role akin to services in SOA [39, 43]. For successful implementation of this type of system, [39] proposes three requirements that should be satisfied:

- *Extensibility of capability descriptions.* Manufacturing equipment is typically engineered for utilization over extended periods, ranging from 10 to 20 years or even more. Such equipment has to be designed with a focus on later adaptation. This means that mechanical interfaces and software implementations, capability models have to be created in a way that allows for changes at a later point [39].

- *Vendor neutrality and linked information.* Manufacturing facilities frequently incorporate machinery and solutions sourced from various vendors, resulting in the use of diverse tools for implementation. A representation of machines and their capabilities should be stored in an open and vendor-neutral format, facilitating the integration of information from disparate sources and mitigating challenges arising from the mentioned heterogeneity. The presence of standards is crucial for the development of adaptable systems [39, 44].

- *Distinction between abstract and executable capabilities.* A distinct division should be established between outlining a machine's capabilities and detailing the technical solutions available for carrying out these capabilities. Abstract capabilities can also be articulated as a request from a plant operator seeking technical solutions to address a specific issue. Equipment manufacturers can then provide machines equipped with executable functions that fulfill the requirements stipulated by the plant operator [39].

Successful Plug and Produce system implementation hinges on three crucial aspects. First, prioritize extensibility for adapting manufacturing equipment over its extended lifespan. Second, ensure vendor neutrality and linked information to handle diverse machinery and solutions from different vendors, emphasizing open, vendor-neutral formats and standards. Lastly, maintain a clear distinction between abstract and executable capabilities, fostering effective communication between plant operators and equipment manufacturers. These requirements form a solid foundation for implementing the Plug and Produce system in manufacturing environments.

The following project can be observed as an example of a Plug and Produce system implementation. The case study, based on the EnAS demonstrator [45], uses a product-centric control strategy. This approach involves direct product requests to available providers for assembly and processing, eliminating the need for offline planning. The implementation of product-centric control is executed using IEC 61499. Also, the work integrates the SOA paradigm, using OPC UA to dynamically discover and compose production services based on incoming product orders and types [40].

The Open Platform Communications Unified Architecture (OPC UA) Discovery standard is proposed as a service discovery mechanism, as illustrated in Figure 2.9. This mechanism introduces a discovery server where services can register themselves, becoming visible to other parties after initialization and integration into the system. Services are registered to the discovery server through the OPC UA server, and parties with an associated OPC UA client can query the discovery server to identify available services and take appropriate actions based on this information [40, 46].

Another example of a Plug and Produce system is presented in [23]. This work is based on a skill-based approach and presents a concept of how skills implemented

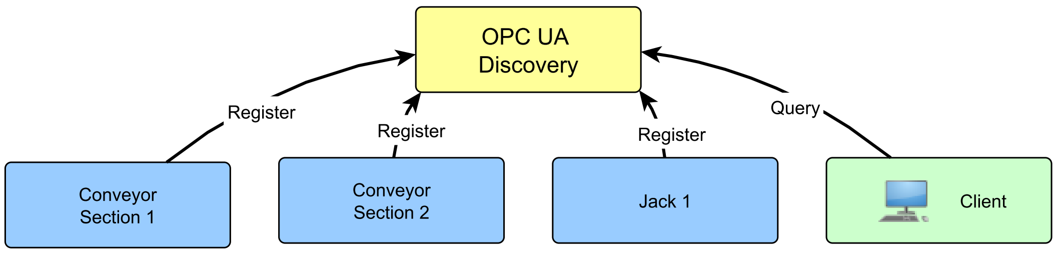


Figure 2.9: The architecture that utilizes OPC UA Discovery for service discovery [40].

with different standards can be managed by a common interface of middleware software. A common interface guarantees to the services easy communication and control of the execution process of services. Authors assume the terms service and skill as equivalent terms. The common interface is represented by OPC UA and establishes communication between upper control levels and devices on the field to allow a SOA-oriented design for the system. Skills are implemented with both IEC 61131-3 and IEC 61499 and mapped to OPC UA [23].

A setup consists of three color lamps controlled by three different PLCs. Two light controls are implemented in IEC 61499 and the last one in IEC 61131-3 [23]. Each atomic skill, which turns the corresponding lamps on and off, is represented in the OPC UA namespace that uses a program finite state machine as shown in Figure 2.10. In OPC UA, a Finite State Machine (FSM) models the behavior of components, specifying states, transitions, and associated actions in response to events or conditions. It aids in describing and controlling dynamic behaviors in industrial automation [47]. The OPC UA client manages a program by calling methods and allows not just to trigger a skill's execution, but also to receive a result back and monitor intermediate results. A server runs skills and can prevent a method's execution by a client if it is now allowed in the current state. Utilizing an OPC UA interface as a standardized skill description provides an abstraction of low-level functionality implementation, showcasing interoperability across diverse software implementations on different devices [23, 48].

As it was described before *Skills* are an essential part of the Plug and Produce approach. In practical realization, the IEC 61499 standard is the most widespread way for Skill implementation. IEC 61499 is proposed by [44], [49] and also used in projects like EnAS [40, 45], openMOS [50], IDEAS [25]. [23] shows that the standard IEC 61131-3 is a possible and alternative solution for skills implementation as well.

OPC UA programs which are used in [40], [46], and [23] provide the functionality for detecting components of a system within a network. This feature gives a

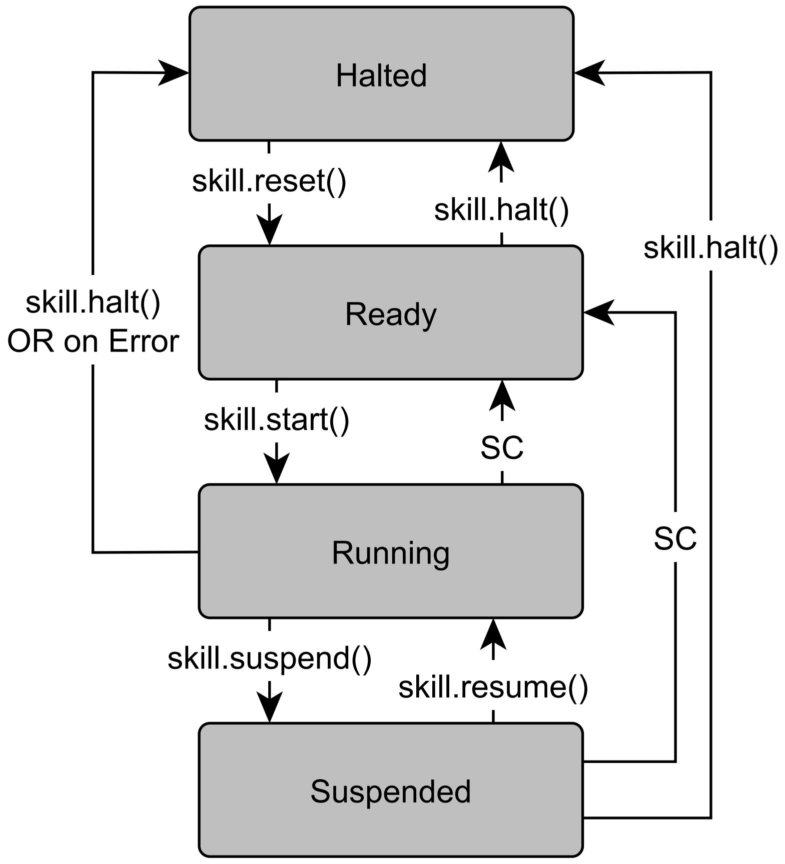


Figure 2.10: A Skill modeled as an OPC UA program finite state machine. Atomic skill is represented in the OPC UA namespace with five methods (Start, Suspend, Resume, Halt, Reset) and state variable. *Start()* turns a lamp on, *cancel()* turns a lamp off. *Reset()*, *resume()*, and *suspend()* have no effect when calling. This modeling is fully compliant with the OPC UA programs specification. All control methods are defined as optional [23, 48].

possibility to build the Plug and Produce system architecture based on OPC UA.

An additional tool, proposed in numerous papers, that can enhance and simplify the development of an OPC UA information model is Automation Markup Language (AutomationML) [51, 52]. The primary objective of AutomationML is to integrate engineering tools across diverse domains, including process control engineering, mechanical plant engineering, electrical design, process engineering, robot programming, PLC programming, HMI development, and more [44, 49]. It aims to address the bottleneck in data exchange arising from the use of various proprietary data types and formats in manufacturing plants. [53]. When combined, AutomationML and OPC UA enable easy integration and interoperability between different devices and systems in a Plug and Produce environment. AutomationML can describe the engineering details of machines, and OPC UA can facilitate the communication and interaction between these machines by providing a standardized communication platform. This integration ensures that devices from various vendors can work together harmoniously, promoting flexibility and efficiency in industrial processes [49, 51, 52].

The Plug and Produce concept is crucial for Industry 4.0 because it addresses the need for flexible and adaptable manufacturing systems. With shorter product life cycles and demand for customized products, the ability to quickly and effortlessly integrate new equipment is essential. Plug and Produce enables manufacturers to achieve dynamic reconfigurability, adaptability, and efficient commissioning, ultimately contributing to increased operational efficiency and responsiveness to changing market demands in the Industry 4.0 landscape.

## 2.4 Search Algorithms

The shortest path algorithms are invaluable tools in the industrial sector. Their applications range from optimizing logistics to robotic navigation improving the overall performance of industrial processes. A pipe system is a great example of an application field. The complex pipe's network looks from the top view similar to a graph. That means the graph theory can be used to represent the pipe's network. The point, that the search for a shortest path is a classical problem in a graph theory, makes it even more convenient and suitable way to solve this problem.

### 2.4.1 Graph

Formally, a *graph* is a pair of sets $(V, E)$, where $V$ is the set of vertices and $E$ is the set of edges, connecting the pairs of vertices [54]. Graphs can be classified into several types based on their characteristics and properties. For example, an *undirected* graph is a type of graph in which edges between nodes (vertices) have no direction. The relationship between two nodes is mutual, and it doesn't matter which node is considered first. In a *directed* graph each edge has a direction, which means it goes from one vertex to another. These edges are represented as arrows, and there can be one-way or two-way connections between vertices. Another type of graph is a *weighted* graph is a graph in which each edge has an associated numerical value called a weight. These weights represent some measure of distance, cost, or importance between the connected nodes. In contrast to weighted graphs, *unweighted* graphs have edges with no associated values or weights [22, 55].

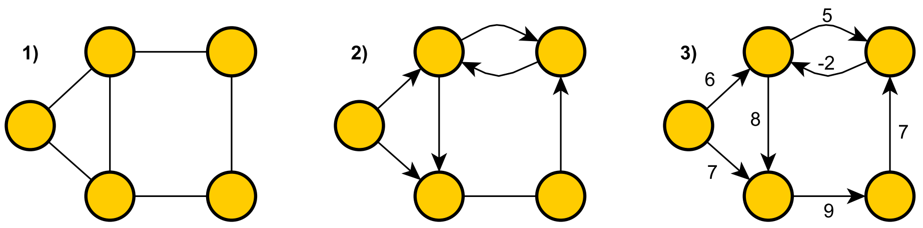


Figure 2.11: Different types of graphs. Picture 1 illustrates the undirected and unweighted graph, where circles are nodes (vertices) and connected with edges. Picture 2 is the directed but unweighted graph. Picture 3 is the directed and weighted graph.

Graphs allow to solve optimization problems, design, and search for an optimal route between a start node and a goal node. A selected path on a graph is defined by a search strategy while the search strategy is defined by the graph's complexity and possible, addition requirements to a task. If the node at the end of the selected path is not a goal node and it has no neighbors, then extending the path means removing the path from the frontier. This outcome is reasonable because this path could not be part of a path from the start node to a goal node [22].

## 2.4.2 Dijkstra's algorithm

The Dijkstra shortest path algorithm was developed by Dutch computer scientist Edsger W. Dijkstra in 1956. It is a widely used and efficient algorithm for finding the shortest path between two nodes in a weighted, directed graph with non-negative edge weights and guarantees finding the shortest path from the source node to all other nodes in the graph. However, it does not work with graphs that have negative edge weights. The Dijkstra algorithm has various practical applications [56, 57]:

- Determining the shortest path for data packets in computer networks;

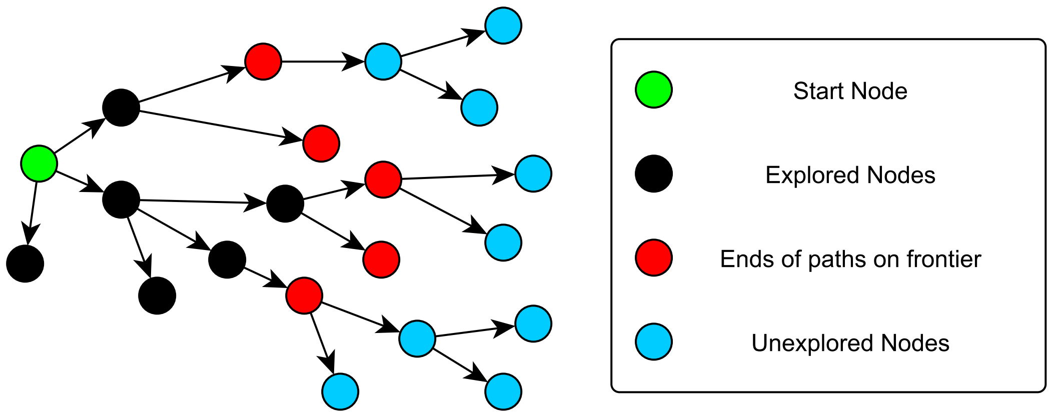


Figure 2.12: Problem solving by graph searching. The intuitive idea behind the generic search algorithm, given a *graph*, a *start node*, and a *goal* predicate, is to explore paths incrementally from the start node. This is done by maintaining a *frontier* of paths from the start node [22].

- Calculating optimal routes for drivers based on road network data;
- Planning efficient routes for public transportation or delivery services.

The Dijkstra algorithm works by maintaining a set of tentative distances from a source node to all other nodes in the graph. It repeatedly selects the node with the shortest tentative distance and explores its neighboring nodes, updating their distances if a shorter path is found. This process continues until the shortest path to all nodes is determined [56, 57].

Execution of the Dijkstra Algorithm is presented in Figure 2.13 and has following steps:

- Step 1: Initialize the distance from the source node to itself as 0 and the distance to all other nodes as infinity. Create an empty set of visited nodes.
- Step 2: Select the node with the smallest tentative distance (initially the source node) and mark it as visited.
- Step 3: For the selected node, examine all its unvisited neighbors. Calculate their tentative distances through the current node, and update the distances if a shorter path is found.
- Step 4: Repeat step 2 until all nodes have been visited or the target node is reached [56, 57].

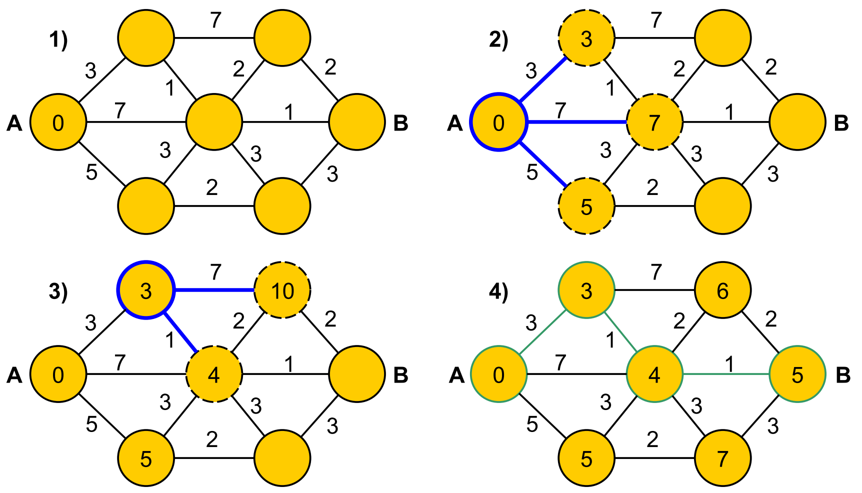


Figure 2.13: Execution of the Dijkstra algorithm. Picture 1 shows the initialization of the distance from the source node to itself as 0. Picture 2 is the selection of the node with the smallest tentative distance and examination of all its unvisited neighbors. Picture 3 is a repetition of the previous steps. Picture 4 is the shortest path from A to B.

The time complexity of the Dijkstra algorithm is $O(V^2)$ without a priority queue, where V is the number of nodes. With a priority queue, the time complexity is reduced to $O(V * log(V) + E)$, where E is the number of edges. The latter case is more efficient and suitable for larger graphs [56, 57].

### 2.4.3 Bellman-Ford algorithm

The Bellman-Ford algorithm is a well-known algorithm for finding the shortest paths in a weighted directed graph and works for graphs with both positive and negative edge weights. This algorithm operates by iteratively relaxing the edges of the graph to find the shortest paths from a source node to all other nodes. One of the features of the Bellman-Ford algorithm is its ability to detect the presence of negative weight cycles. Negative weight cycles can lead to arbitrarily small distances and make the problem of finding the shortest path ill-defined. The usage of the Bellman-Ford algorithm can be observed in such applications as [56, 57]:

- Finding the shortest path in computer networks;
- Determining optimal routes for delivery trucks or public transportation;
- Analyzing financial networks and risk assessment.

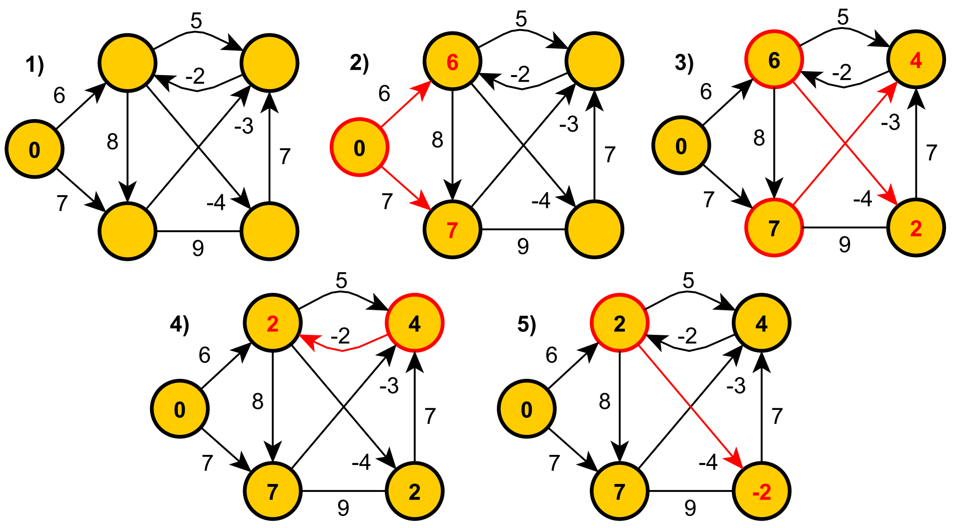


Figure 2.14: Execution of the Bellman-Ford algorithm. Picture 1 shows distance initialization to itself. Picture 2 is an iteration through all edges. The red number is the updated distance. Pictures 3-5 repeat of previous step.

Execution of the Bellman-Ford algorithm is presented in Figure 2.14 and has following steps:

- Step 1: Initialize the distance from the source node to itself as 0 and the distance to all other nodes as infinity.
- Step 2: Iterate through all edges of the graph, relaxing each edge. Edge relaxation involves checking if the distance to the target node through the current edge is shorter than the previously known distance. If it is, update the distance.
- Step 3: Repeat step 2 for a total of (V-1) iterations, where V is the number of nodes in the graph. This is because the shortest path between any two nodes in a graph with V nodes has at most (V-1) edges.
- Step 4: After (V-1) iterations, if the algorithm finds shorter distances, it means that there are negative weight cycles in the graph, and the algorithm can't guarantee accurate results [56, 57].

The time complexity of the Bellman-Ford algorithm is $O(V * E)$, where V is the number of nodes, and E is the number of edges. It can be less efficient than algorithms like Dijkstra's algorithm for graphs with non-negative weights. However, its ability to handle negative weights and detect negative weight cycles makes it a valuable tool in scenarios where other algorithms may not be suitable [56, 57].

### 2.4.4 Floyd-Warshall algorithm

The Floyd-Warshall algorithm was developed by American computer scientists Robert W. Floyd and Stephen Warshall in the early 1960s. It finds the shortest paths between all pairs of nodes in a weighted directed graph. Unlike Dijkstra's algorithm, the Floyd-Warshall algorithm can handle graphs with both positive and negative edge weights and is capable of detecting negative weight cycles. The algorithm is well-suited for solving problems where a complete pairwise distance matrix is required, such as in-network routing or transportation planning. Fields of application for the Floyd-Warshall algorithm are [56, 57]:

- Determining optimal routes for data packets in computer networks;
- Finding the most efficient routes for transportation networks;
- Measuring distances and relationships between users in a social network.

The Floyd-Warshall algorithm is based on dynamic programming and uses a matrix to store intermediate results. It iteratively updates the matrix to find the shortest path between all pairs of nodes [56, 57]. Execution of the Floyd-Warshall Algorithm has several steps:

- Step 1: Initialize a matrix where each element (i, j) represents the shortest distance from node i to node j. Initially, this matrix is filled with the direct edge weights if an edge exists, or with infinity if there is no direct edge. The diagonal elements (i, i) are set to 0.
- Step 2: Perform a series of iterations. In each iteration, consider a specific node (k) as a potential intermediate node in the paths from each node (i) to each node (j).
- Step 3: For each pair of nodes (i, j), check if the path from i to j through node k is shorter than the current path from i to j. If it is, update the distance matrix with the new, shorter distance.
- Step 4: Repeat Step 2 for all nodes as potential intermediaries (k). After completing all iterations, the distance matrix will contain the shortest distances between all pairs of nodes [56, 57].

The Floyd-Warshall algorithm has a time complexity of $O(V^3)$, where V is the number of nodes in the graph. This makes it less efficient than Dijkstra's algorithm for graphs with non-negative edge weights. However, its ability to handle a wider range of graph types and its capability to compute all-pairs shortest paths in a single run makes it a valuable tool in certain scenarios [56, 57].

### 2.4.5 Johnson's algorithm

Johnson's algorithm is a specialized algorithm for finding the shortest paths between all pairs of nodes in a weighted, directed graph, just like the Floyd-Warshall algorithm. It was developed by American computer scientist Donald B. Johnson in 1977 and offers a more efficient approach for certain cases compared to the Floyd-Warshall algorithm. Johnson's algorithm works for graphs with both positive and negative edge weights and can handle graphs with negative weight cycles. It is especially efficient when the graph contains only a few negative weight edges relative to the total number of edges. Typical examples of Johnson's algorithm usage are [58, 59]:

- Determining the shortest path for data packets in computer networks;
- Measuring distances and relationships between users in social networks;
- Identifying important nodes or centralities within a graph.

The Johnson algorithm is based on the idea of transforming the original graph with potentially negative edge weights into a modified graph with non-negative edge weights. This transformation allows the use of a faster algorithm (basically Dijkstra's algorithm) for finding the shortest paths between all pairs of nodes [58, 59].

Execution of the Johnson's algorithm is presented in Figure 2.15 and has following steps:

- Step 1: Create a new, temporary source node, and connect it to all other nodes in the graph with zero-weight edges.
- Step 2: Run the Johnson's algorithm starting from the source node to calculate the shortest paths from the source node to all other nodes. This step detects and eliminates negative weight cycles in the graph.
- Step 3: After the Johnson's step, edge weights in the original graph should be reassigned by removing the source node and using the calculated distances from the source node to adjust the weights in the form: $Newweight = originalweight + startnode - endnode$.

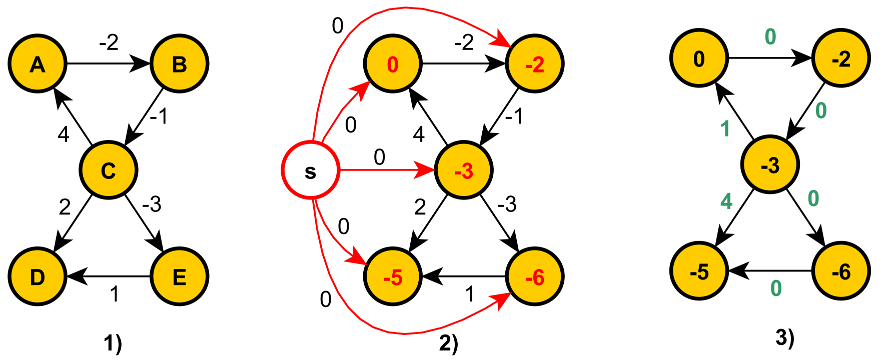


Figure 2.15: Execution of the Johnson's algorithm. Picture 1 shows the original graph. Picture 2 illustrates the temporary source node and calculated distances from this node to all other nodes in the graph. Picture 3 shows the graph with recalculated weights, e.g. for the edge AB the new weight is: -2+0-(-2)= 0 $(Newweight = originalweight + startnode - endnode)$.

- Step 4: For each node in the modified graph, Dijkstra's algorithm is used to find the shortest paths to all other nodes. This can be done efficiently because the modified graph has non-negative edge weights [55, 58].

The time complexity of Johnson's algorithm mainly depends on the underlying algorithm used for the shortest path calculations, typically Dijkstra's algorithm. In the worst case, Johnson's algorithm has a complexity of $O(V^2 * log(V) + VE)$, where V is the number of nodes and E is the number of edges. It is more efficient than the Floyd-Warshall algorithm for graphs with relatively few negative weight edges [55, 58].

### 2.4.6 A* search

The A* algorithm is an efficient pathfinding algorithm commonly employed in graph traversal and pathfinding problems. It's particularly effective for finding the shortest path between two points in a graph or grid. This algorithm was developed by American computer scientists Peter Hart, Nils Nilsson, and Bertram Raphael in 1968, and is used in robotics, GPS navigation, and other fields [21, 22, 58].

A* contains elements of the Dijkstra's algorithm and uses a heuristic to estimate the cost of reaching the goal from each node, making it a more informed search than Dijkstra's algorithm. The algorithm maintains a priority queue of nodes to explore and selects the next node to visit based on a cost function that balances both actual path cost and estimated remaining cost [21, 22, 58].

The key component of the A* algorithm is a cost function. It evaluates each node based on a cost function *f(n)*, which is the sum of two components: *f(n) = g(n) + h(n)*. The first is the actual cost of the path *g(n)* from the start node to node n. The second is the heuristic function *h(n)* estimate of the cost from node n to the goal. The choice of heuristic can significantly impact the algorithm's performance [21, 22, 58].

The time complexity of A* is highly dependent on the choice of heuristic and the specific problem instance. In the worst case, the time complexity is exponential. However, A* is usually very efficient in practice, especially when a good heuristic is available. To summarize the previous chapter, the right choice of the shortest path algorithm is crucial and depends on the specific characteristics of the task. It involves considering the graph type, edge weights, presence of negative weights, and heuristics. There is no universal solution, and the choice of the shortest path algorithm should be tailored to the unique characteristics and demands of the problem.

## 2.5 IEC 61512 / ISA-88

Today, the entire modern industry operates under the auspices of technological standards. In the process industry ANSI/ISA S88 and its equivalent IEC 61512 provide domain specific models for the design and control of batch production processes, as well as describes the norms for recipes of the technological process [60].

### 2.5.1 Batch process

Industrial processes can be categorized as continuous, discrete, or batch operations. In brief, *continuous processes* involve a continuous outflow, such as energy, paper, or steel production. *Discrete processes* are characterized by distinct, individually identifiable outputs, e.g. car manufacturing, where each car and its components can be tracked separately. *Batch processes* combine attributes of both continuous and discrete processes, resulting in batch outcomes, and managing such processes is known as Batch Control [61–64].

Batch processes are an integral component of the chemical process industries and find widespread use in the production of top-quality goods. Their widespread adoption can be primarily attributed to their flexibility in managing diverse product grades by altering initial conditions and input trajectories. Generally, a batch process comprises the following primary stages [65]:

- The reactor is filled with specified ingredients as per a predetermined recipe.

- A transformation process is executed over a finite time interval.
- Ultimately, the process concludes upon satisfying specific predefined criteria.

In the early 1990s, a standard known as ISA S88 was developed and published, with a primary focus on batch processes and their control. This standard introduces terminology and concepts that simplify the design and operation of batch plants [61]. The core idea of ISA 88 is the separation of product knowledge from the equipment. The standard proposes a set of seven models as shown in Figure 2.16 to describe a batch process in varying levels of granularity from process and control engineering points of view [66].

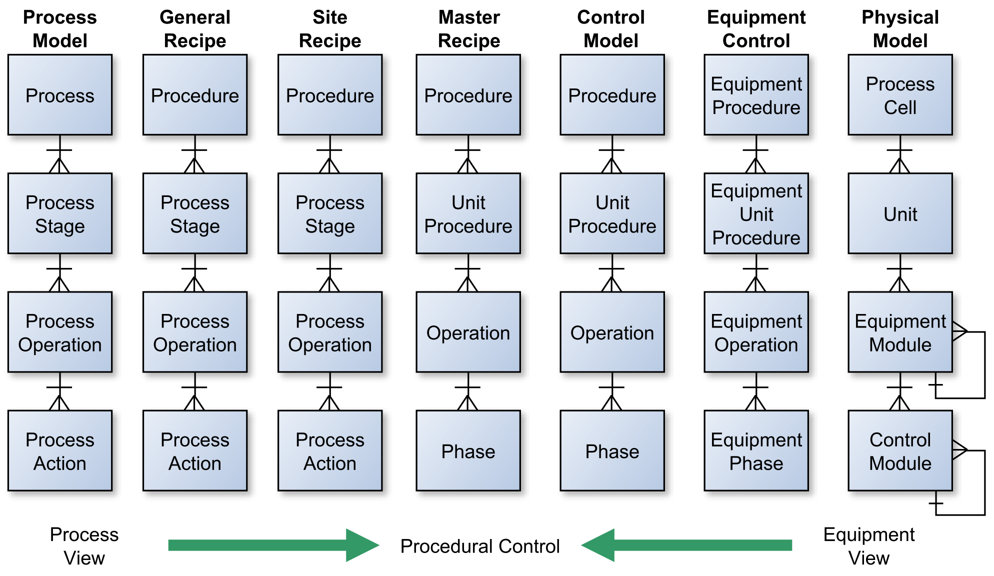


Figure 2.16: Process and equipment view of IEC 61512/ISA 88 [66].

From the process view, planning starts with a *Process model*. The *Process model* encompasses the dynamic and functional behavior of the batch process. It describes how the materials and substances are transformed, mixed, reacted, or otherwise processed as they move through the equipment defined in the Physical Model. The *Process Model* focuses on the key parameters, conditions, and control strategies that govern the actual process dynamics. It is concerned with variables like temperature, pressure, flow rates, and chemical reactions. The *Process Model* helps in understanding and controlling the real-time behavior of the batch process [62, 66–68].

The transition from a general recipe to a specific control recipe is progressively refined to align with the batch plant's requirements. The resulting *Procedural* *control model* focuses on the sequencing and organization of actions, tasks, and procedures that need to be carried out to execute a batch process. It defines the order in which specific actions and unit procedures must occur to produce a product or achieve a desired outcome. The *Procedural Model* outlines the recipe structure, which includes master recipes, product recipes, and process recipes. It provides a step-by-step description of how the process should be executed, including the allocation of resources and timing of actions [62, 66–68].

The *Physical model* refers to the physical equipment, devices, and components involved in a batch process. It defines the actual physical assets, such as tanks, pumps, valves, sensors, and other equipment, that are used in the manufacturing or production process. The *Physical Model* describes the hardware and instrumentation that interact with the substances and materials being processed. It serves as the foundation for understanding the layout and configuration of the industrial plant or system [62, 66–68].

These three models are interrelated and collectively form the foundation for designing, simulating, and controlling batch processes. The Physical Model defines the equipment, the Procedural Model outlines the sequence of actions, and the Process Model addresses how the substances are transformed during the process. Together, they provide a comprehensive framework for developing and managing batch control systems [62, 66–68].

The standard was originally developed with a specific focus on batch processes. However, its applicability extends to continuous and discrete processes that demand a degree of flexibility. It provides a standardized terminology aimed at enhancing communication to prevent situations where different individuals use varying terms for the same concept or, conversely, employ the same term for distinct concepts. Furthermore, an essential facet of ISA S88 is its emphasis on modularity. This modularity allows for the development of process equipment and procedures that can be reused across various applications [61, 64].

### 2.5.2 Classification by physical structure

The IEC 61512 standard provides a framework for designing and modeling batch processes and allows to choose the most appropriate structure based on the complexity and requirements of the specific process. This choice impacts how the batch recipe is executed, monitored, and controlled, making it a critical consideration in batch control system design.

**A single-path structure** represents a straightforward, sequential approach to executing a batch process (see Figure 2.17). This structure has a single, predetermined path that defines the sequence of actions and operations to be performed. The process follows a linear and unidirectional flow, with each step occurring one after the other. It is often used for simple batch processes with well-defined and fixed procedures [62].

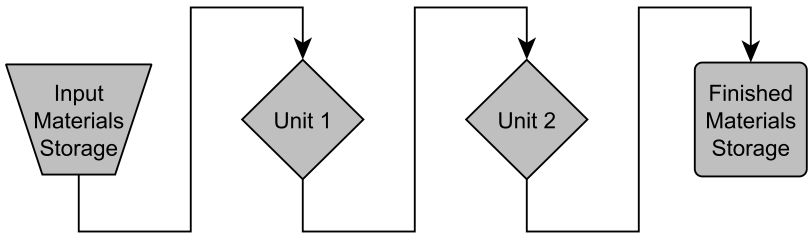


Figure 2.17: Single-path structure. It represents a group (or a single unit) of sequentially connected units and has a single, predetermined path that defines the sequence of actions and operations to be performed [62].

**A multiple-path structure** is shown in Figure 2.18. It allows for branching and the execution of different paths or alternatives within a batch process. This means that at certain decision points or conditions, the process can follow different routes, each with its own set of actions or operations. Multiple-path structures are useful for batch processes that require flexibility, adaptive decision-making, or handling of various scenarios [62].

**A network structure** is shown in Figure 2.19. It is the most complex and flexible of the three. It allows for the creation of a network of interconnected paths and actions within a batch process. In this structure, different unit procedures, operations, and phases can be linked in a non-linear, interdependent manner. The network structure is highly versatile and suitable for complex batch processes that involve intricate interdependencies, parallel operations, or simultaneous execution of multiple tasks [62].

## 2.5.3 Recipes

A recipe, in the context of manufacturing, serves as a comprehensive set of instructions that precisely defines the production requirements for a particular product. This is an analog of a cooking recipe but designed for the production of goods. Within the framework of the IEC 61512 standard, four distinct recipe types are discussed: the general recipe, site recipe, master recipe, and control recipe (see Figure 2.20). Each of these recipe types contains specific information related to the production process of a particular product. The significance of these recipes lies in their ability to enable batch processing equipment to produce various products without necessitating the reconfiguration of equipment controls for each product. In other words, these recipes streamline the manufacturing process by providing a standardized blueprint for creating multiple products efficiently [61, 62].

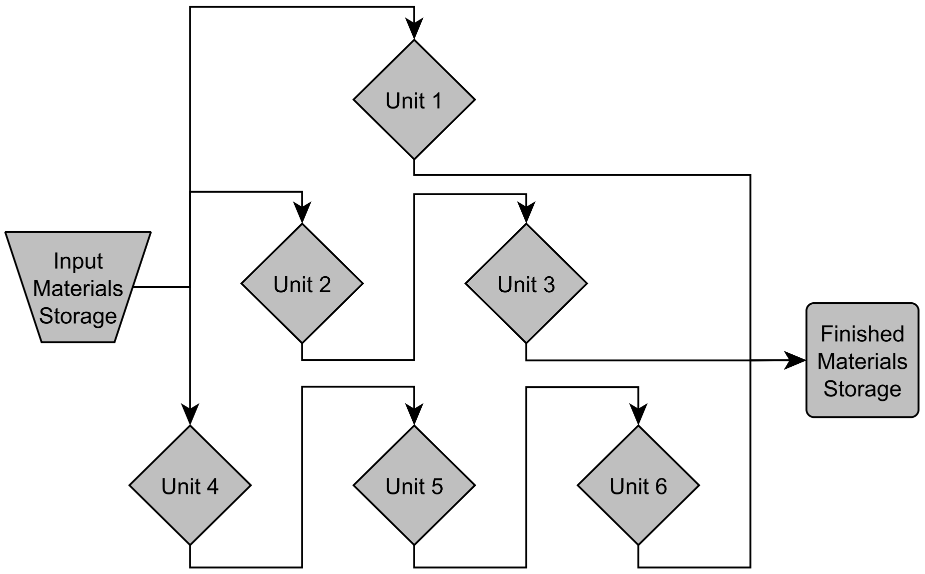


Figure 2.18: Multiple-path structure. The process can follow different routes, each with its own set of actions or operations. The units can share raw materials and product storage. Several batches may be in progress at the same time. [62].

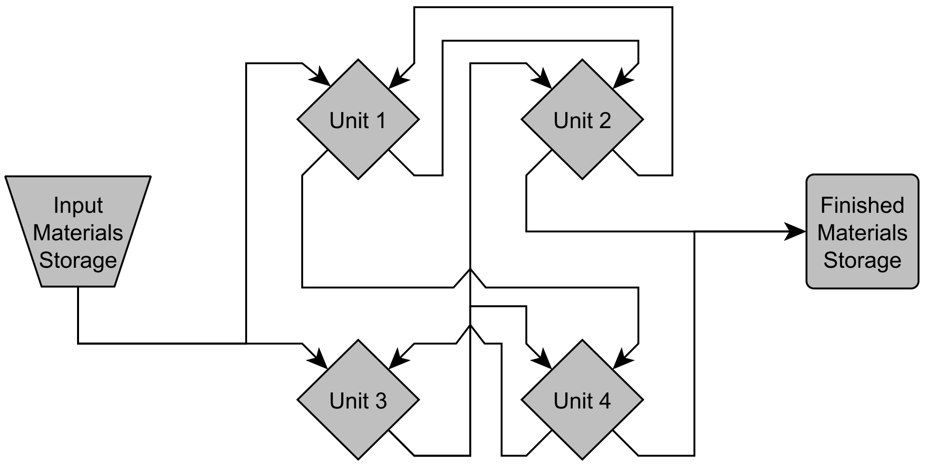


Figure 2.19: Network structure. The most complex and flexible structure that allows for the creation of a network of interconnected paths and actions within a batch process [62].

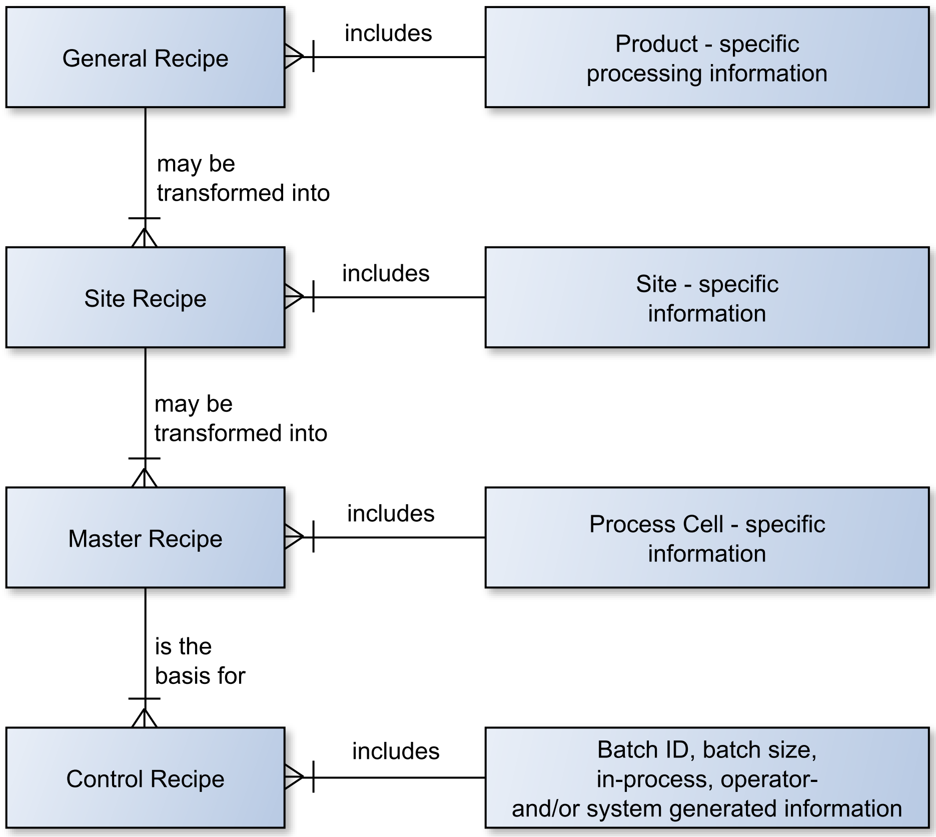

General Recipe
includes
Product - specific processing information
may be transformed into
Site Recipe
includes
Site - specific information
may be transformed into
Master Recipe
includes
Process Cell - specific information
is the basis for
Control Recipe
includes
Batch ID, batch size, in-process, operator- and/or system generated information

Figure 2.20: ISA-88 Recipe Model. The general recipe is a basis for lower-level recipes and not equipment specific. The site recipe is a combination of site-specific information and a general recipe. The master recipe is equipment and product-specific. It is targeted to a process cell or a subset of the process cell equipment but not specific to batch. The control recipe is specific to a single batch and contains scheduling and operational information [62].

The *general recipe* serves as the foundational recipe for lower-level recipes. It is designed without specific knowledge of the production equipment employed for manufacturing the product. The general recipe outlines the raw materials, their relative quantities, and the necessary processing steps, all without tying them to a particular facility or its available equipment [62, 69].

The *site recipe* is adapted to a particular production facility, combining facility-specific information with the enterprise-level recipe. Typically derived from a general recipe recipe, it customizes the conditions to meet the requirements of a specific production site. This level of detail is essential for facility-specific, long-term production planning but doesn't specify the particular equipment combinations. Multiple site recipes can be derived from the same general recipe, each encompassing a part of the process that can be implemented in a particular facility [62, 69].

The *master recipe* details the process engineering implementation of a process outlined in the general recipe. It requires precise information about the process flow, production scale, equipment types, and the level of automation. Multiple process implementation recipes can be derived from a site recipe, each encompassing a part of the site recipe that can be executed in a specific unit. This level includes all the data from the site recipe, along with production instructions and procedural requirements for specific process segments. The master recipe is no longer equipment-neutral because it contains process and automation details about the process flow [62, 69].

The *control recipe* is created from the master recipe, the control recipe is generated based on the requirements for executing production on a specific production facility. It includes order-specific information like production dates, quantities, material batches, and batch numbers, as well as details about planned start dates, duration, and subunit assignments. The control recipe also incorporates all data and values relating to quantity, quality, and the progression of the process [62, 69].

The second part of the standard, IEC 61512 Batch Control Part 2: Data Structures and Guidelines for Languages [70], present Procedure Function Chart (PFC). PFC is a graphical representation used to define, document, and illustrate the procedural elements and control logic within a batch process. It provides a visual overview of how a batch procedure is structured and executed. It includes various graphical symbols and annotations to depict the different components and elements of a batch procedure, such as unit procedures, control functions, sequences, and the relationships between them [66, 68, 70]. Figure 2.21 as an example of PFC from [66] and table of graphical elements.

*Procedures* are the highest-level set of instructions for a batch process, consisting of various *Unit Procedures*. *Unit Procedures*, in turn, are composed of *Operations*, and *Operations* can be further broken down into *Phases*. This hierarchical structure allows for a systematic and detailed representation of batch processes, making it

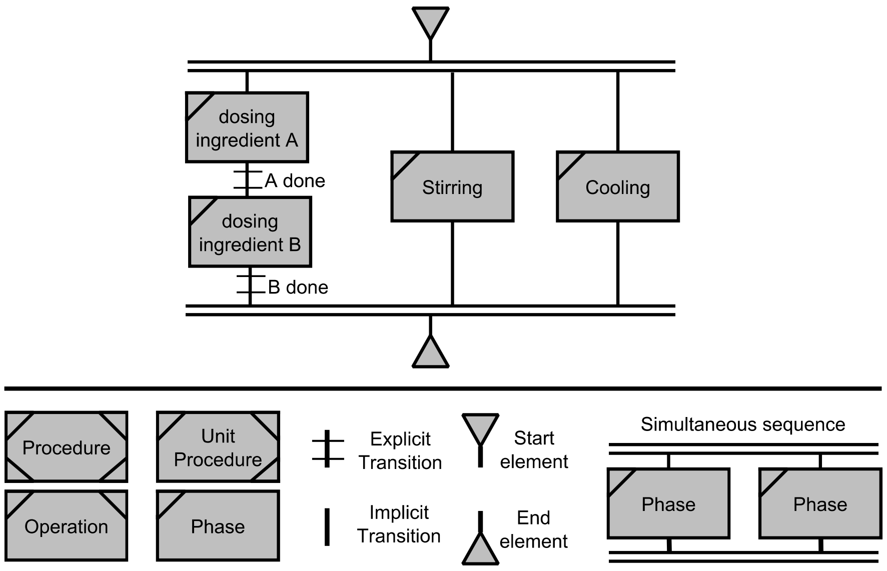


Figure 2.21: Elements of PFC. The top image shows an example of implementation for a filling operation of a reactor adapted from [66]. The image below illustrates different graphical elements of PFC from IEC 61512.

easier to control and execute complex industrial operations [62].

PFC serves as a valuable tool for operators, engineers, and other stakeholders involved in batch processes, as it offers a clear and intuitive representation of the sequence of actions and decision points within a batch recipe. The main purpose of PFC is to make batch procedures more understandable and manageable, enabling efficient execution and control of batch processes. It plays a crucial role in ensuring that batch operations are carried out accurately and consistently, in line with the defined procedures and control strategies [66, 68, 70].

## 2.6 OPC Unified Architecture (OPC UA)

The Industry 4.0 and the Internet of Things brought various new concepts. In scenarios covering a wide array of application domains and their requirements, it is hard to name a single network protocol for every situation. An attempt to choose a single communication protocol reduces the flexibility of a system and cuts off the possibility of using the benefits of a certain protocol [71, 72].

Effective communication within the context of Industry 4.0 requires flexible and adaptable connections between integration layers and information layers. To avoid this lack of flexibility a useful multi-protocol communication protocol, which can help to solve a compatibility problem, should be selected [71]. OPC UA is a potential candidate to be one of the key standards within the context of Industry 4.0. It helps to solve the problem of data accessibility between industrial controllers from different vendors and ensures compatibility with a wide range of embedded devices [73].

With OPC UA, all data and resources are accessible to every authorized application at any time in the address space (collection of visible information on a Server [74]). This function is independent of the manufacturer from which the applications originate, the programming language in which they were developed, or the operating system on which they are used [75].

An example of enhanced flexibility with the help of OPC UA is presented in [76]. The topic focused on the migration of legacy systems in an industry into compatible with Industry 4.0 systems. A legacy RFID system consists of four independent stations. Each contains an industry robot, PLC, actuators, and sensors. Usage of Industry 4.0 standards such as OPC UA combined with a cloud computing platform (Microsoft Azure) helps to provide the legacy system with a digital interface. The authors propose that the implemented architecture has modular properties and easily can be extended with additional equipment (i.e. antennas, PLCs). Another example of communication between devices from various vendors is presented in [73]. The paper proposes an OPC UA client/gateway-based architecture that is implemented as a multiple OPC UA server connection to a single OPC UA client (gateway) device. The OPC UA client connects three different components: cloud services (Microsoft Azure and Amazon Web Services), PLCs, and embedded system controllers (Raspberry Pi controller). This architecture is aimed to simplify smart factories design and give engineers additional flexibility during the system's components choice. Both [76] and [73] works show how the OPC UA protocol can help traditional PLCs be more compatible with Industry 4.0 standards.

An approach proposed in [71] shows how can be achieved interoperability between OPC UA and SOA protocols. This provides integration of IoT devices with legacy automation systems. In this approach, the OPC UA is used as a translator and to perform the role of a service. It works with standard IoT protocols (i.e. HyperText Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), and Message Queuing Telemetry Transport (MQTT)) and allows access from non-OPC UA-based IoT applications to OPC UA nodes [71]. A concept in [77] shows an attempt to achieve the advantages of both protocols by extending OPC UA with MQTT. The semantic data model of OPC UA is extended with MQTT in accordance with the OPC UA publish/subscribe framework, allowing for broker-based data exchange. Addressing the divergent messaging concepts of OPC UA and MQTT, this work successfully resolves the data mapping challenge between these two protocols. These two works give an idea of OPC UA usage in tandem with other common protocols to enhance the flexibility of a system.

OPC UA is the interoperability standard that provides the secure and reliable exchange of data in the industrial automation domain. It is platform-independent and provides a seamless data flow between devices from multiple vendors [75]. OPC UA is developed by the OPC Foundation and standardized in IEC 62541. The OPC Foundation is also responsible for the development and maintenance of this standard [75, 78]. It outlines services that OPC UA clients use for interaction with information models maintained on the server [71].

OPC UA is extensively utilized in industrial applications for connecting equipment across various networks and at different tiers of the automation hierarchy. This protocol offers object-oriented data modeling and organizes the address space of these objects within the server [72, 79]. It allows the creation of variables of certain data types within each object on the server, allowing clients to read variable values or to subscribe to variables for which they would like to receive updated values [71].

The fundamental element of the information model is a *Node*, an object-oriented entity. Each node is unique and is identified by a *Nodeid*, which contains a *Namespace index* and a *Node name*. Relationships between nodes are established through references, creating meaningful associations. Standardized interactions between an OPC UA client and server are facilitated by a suite of server-provided services. These services enable access to and control of nodes, encompassing tasks such as node and information model management, data reading and writing (including query and subscription-based operations), and the establishment of communication channels for subsequent requests [71].

Client-server communication follows the design paradigm of service-oriented architecture (SOA), with which a service provider receives requests, processes them, and sends the results back with the response [80, 81].

The Publish-Subscribe model (PubSub according to [82]) provides an alternative mechanism for data and event notification. It has been optimized for many-to-many configurations, while in Client-Server communication each notification is for a single client with guaranteed delivery [80, 83]. It provides easier communication between two or more devices by organizing data exchange using nodes. A Subscriber is an equipment or service which is interested in receiving the data. A publisher is an equipment or service that wishes to send or publish information in the topic. Published messages distribution to subscribers equipment are done via the server [72]. With the Publish-Subscribe model, OPC UA applications do not directly exchange requests and responses. Publishers send messages to a Message Oriented Middleware without any knowledge about Subscribers which may or may be not there. Subscribers express interest in specific types of data, and process messages that contain this data, with no need to know where it came from [75, 82].

## 2.7 Research questions

The modern world and the technologies it encompasses change at an incredible speed. To meet customer requirements, industries face the challenge of producing personalized products with small batch sizes. The solution lies in flexible plants that can be quickly updated or reconfigured.

Taking a step in this direction, the standard IEC 61512 was analyzed. It provides a strong theoretical basis for batch processes while not imposing strict rules for their implementation. A combination of the Skill Engineering Model and Service Oriented Architecture presents promising candidates for implementing a batch process, as these approaches share a common feature with the standard - breaking down a complex task into a sequence of simple actions. OPC UA, as a communication tool, should complement and enhance flexibility, providing an element of Plug and Produce capabilities.

As a result, this thesis seeks to find answers to the following questions:

| **Question 1:** |
|---|
| What kind of advantage/disadvantage has detailed refining of an action into a sequence of atomic skills over a more common skill description? |

| **Question 2:** |
|---|
| Does this approach potentially able to increase the flexibility of the plant? |

# CHAPTER 3

# Concept overview

This chapter gives a general overview of a system as well as possible solutions for its implementation for automatic planning of a batch process. The concept also should be able to provide an operator with a suitable interface for monitoring the state of the process and give alarming functions to prevent accidents during process execution. The concept description may contain some specific details, but it should be suitable for different types of domains with some minor changes. The goal of the chapter is to present a system that can be easily adapted to different industries and types of processes while ensuring safety and reliability of the plant.

## 3.1 Problem statement

The task is to develop a concept for a batch process that allows to run and modify the process. The graphical interface should provide monitoring of the current state, storage of information, as well as manual control of the process. The process includes pouring liquids from containers, mixing, and heating them. For more flexible operation, the system must support the parallel execution of some stages of the process. To ensure stable and safe operation, the system should have an alarming potential.

## 3.2 Basic overview

This concept is based on the ISA-88/IEC 61512 - Batch control standard, which emphasizes the separation of physical equipment from production instructions in batch processes. While the standard does not provide specific guidance on its implementation (only recommendations), this separation is a key element in achieving flexible batch process control. Furthermore, a crucial feature of the standard is modularity, allowing process equipment and procedures to be easily reused in different applications.

Skill-based Engineering Model has the same goals and is, therefore, an excellent candidate for implementing batch processes. In SEM, skill represents actions requested by the recipe processing algorithm and must include all necessary parameters. SEM also allows the implementation of skills independent from an equipment level. The second chapter shows examples of skills implementation using different programming languages and various PLCs from different vendors within one project.

Figure 3.1 shows a scheme of the concept where a field level consists of sensors and actuators as well as PLCs. A PLC has the only task of controlling actuators and collecting data from sensors. The rest of the routine such as batch recipe processing, skills execution, data storage, visualization, and operator control are tasks of an orchestrator (supervising computer) on the control level.

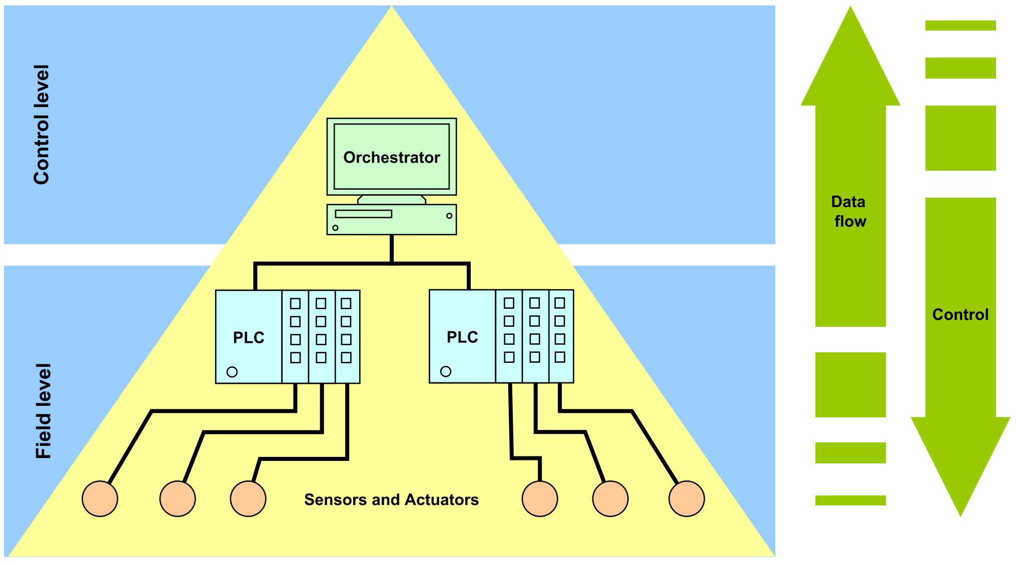


Figure 3.1: The scheme of the concept. The control level is represented as an orchestrator (supervising computer). The orchestrator performs recipe processing, skills execution, and user interface (containing manual control, data storage, and data visualization). The field level consists of sensors, actuators, and PLCs to read sensors and control actuators.

## 3.3 Skill-based approach

Understanding a process is a key point for a recipe implementation. The task considered in this concept can be formulated as follows: *To become a batch of liquid C, X ml of liquid A should be mixed with X ml of liquid B. This mixture should be uniformly heated up to X degree Celsius.* It is obvious that the task consists of 4 specific actions such as:

- Adding process to transfer some volume of liquid from one place to another(two times);
- Mixing process to get a uniform mix ;
- Heating process.

By knowing this information and all necessary input parameters the standard IEC 61512 - Batch control[62] allows to design a recipe for a batch process. For the recipe design, a Procedure Function Chart (PFC)[70] is used. It is a modeling technique that provides a visual representation of the sequence and logic of various functions or steps involved in a specific procedure or process. Figure 3.2 shows the recipe itself. The recipe consists of a set of phases and conditions for their successful execution. However, the recipe is just an instruction that shows how the batch process should be executed.

To implement this recipe as an executable algorithm a skill-based approach is used. As mentioned in the previous chapter it is a method of designing and implementing automation systems that focuses on breaking down the functionality of the system into smaller, reusable building blocks called skills. These skills are function blocks that represent specific functions or tasks that can be performed by the system, such as controlling a motor or measuring a temperature.

Atomic skills provide a basic level of functionality and are used to perform simple tasks (e.g. opening or closing a valve). Composite skills are composed of several other skills (in theory can include both atomic or other complex skills) and provide a higher level of functionality to perform more complex tasks.

For example, to execute the adding process the system should know an initial tank (from where to take the required liquid), a target tank (where to put the required liquid), and the required amount. A search algorithm of find path skill provides the system with knowledge of the valves that should be opened, but it does not have the functionality to open them. At the same time, the open valve skill has the ability to perform an action in the real world, but it does not know which valve should be opened. The combination of these two atomic skills builds a complex skill with the ability to build a calculated path. An additional skill can calculate the volume of pumped liquid. The heating process is less complex

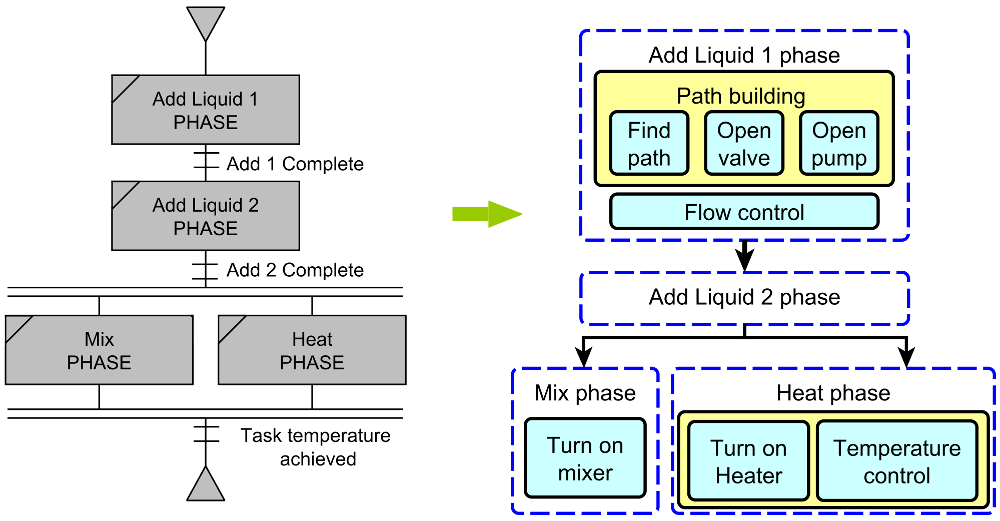


Figure 3.2: Representation of the recipe as a sequence of skills. The left picture is a recipe itself. The right picture shows how the recipe is represented with the help of SEM. The dashed box is one of the recipe phases. The yellow box presents a complex skill, and the blue one is an atomic skill. As it shows each phase consists of one or several skills. Since both add phases have equal functionality the set of skills is presented only for the first add phase.

but also requires several actions. This phase consists of the skill that controls the heater itself and the skill that checks the current temperature in a tank. The mixing process can be represented just as one skill because according to the task, this phase does not contain any specific conditions. The mixer should just work during the whole heating process. Figure 3.2 illustrates a representation of the recipe as a sequence of skills.

The batch process phases are not the only skills for implementation in the system. Section 5 of the standard IEC 61512 - Batch control [62] discusses exception handling in the context of batch manufacturing control. Exceptions are events that deviate from normal or desired behavior. It is essential in batch manufacturing and often makes up a significant portion of control definitions. Examples of events that trigger exception handling include material shortages, equipment problems, and process issues. From a control perspective, handling exceptions involves detecting an event, evaluating it, and generating a response.

Assuming that the plant equipment is known within this concept, several possible exceptions such as responses for them can be identified:

- Tank overfilling. In this case, an operator should receive an alarm notification and the pumping process should be stopped;

- Excessive draining of tanks. This can cause air to enter pipes and damage a pump. This exception has the same response as the previous one;
- Heater protection. The possibility to turn on the heater should be blocked if a liquid level is lower than the heater itself.

## 3.4 Path Planning Algorithm

Before starting to pump a liquid from point A to point B, the system should know how to reach point B. In the case of a redundant pipe system, it is not a trivial task because there are many possible ways with different efficiencies, and poorly designed solutions can even form a loop without an endpoint.

The main task of a searching algorithm is to find the optimal path between an initial and a target tank. The shortest path is not always optimal because it only considers the distance between nodes, but it does not take into account other factors that may affect the overall performance of the system. For example, in a batch processing system, the shortest path may not be the best path in terms of resource utilization. A path that is longer in distance but uses specific resources in the right order may be a better option.

Another factor that could make the shortest path not optimal is the existence of constraints or limitations. Some paths may be shorter in distance but may not be accessible due to technical constraints, such as the capacity of a pipe or the availability of a valve. Under the assumption that the pumping process involves a working pump, it means that the searching algorithm must build a path in a way that includes at least one pump.

Figure 3.3 illustrates the problem statement of the searching path between Tank A and Tank B. The first path *[Tank A - N1 - N2 - Tank B]* has a weight equal to 20, and the second path *[Tank A - N1 - Pump - N2 - Tank B]* has a weight of 24. This means that the first path is the shortest, but this path does to allow to execution of a pumping process. The right choice is to choose the second path for the further process execution even if it is not the shortest, but it is optimal for the current task.

One of the possible ways to force a searching algorithm to build a path through a specific node is the use of negative weights on a graph. Negative weights significantly expand the applicability of shortest path problems as a model for solving other problems. However, it is not enough just to set the weight of the edge for the specific node as a negative number. The choice of negative weight values should align with the current problem. Also, negative weights can create negative-weight cycles in a graph. All this leads to an increase in the complexity of the graph and also increases the time it takes to change it if new elements are added.

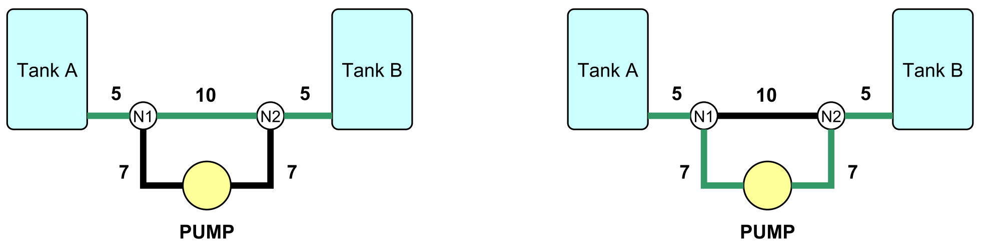


Figure 3.3: Search path problem. The searching algorithm has found two paths between Tank A and Tank B, with the first path being the shortest but not allowing for a pumping process to occur. The second path, although longer, includes a pump and is therefore the optimal choice for the current task of executing a pumping process. This illustrates the importance of considering not only the shortest path but also the specific constraints and requirements of the system when determining the optimal path.

Another way to solve this problem was introduced by M. Baierling in [12]. The idea is to split a desirable path into two subpaths with one common node, which contains a pump. First, the algorithm searches the shortest path from the initial node to the desirable pump, then does the same for the path from the pump to the target node. If a system contains more than one pump this procedure should be executed for each pump. In this case, a path with the lowest sum of two subpaths is the shortest. This approach uses only nonnegative weights, is simple to implement, and makes it easy to change the graph. As was described in the previous chapter Dijkstra's Algorithm [55] is the best candidate for graphs with non-negative edge weights to find the shortest path.

## 3.5 Graphical user interface

A Graphical User Interface (GUI) is a mandatory attribute of any management system. The visual aspect makes it easier for operators to interpret data. While a GUI is valuable for visualizing data and making manual inputs, it often falls short in addressing the complexities of batch process control. Batch processes involve multiple stages, intricate sequences, and the need for real-time monitoring, alarming, and data acquisition. Supervisory Control and Data Acquisition (SCADA) goes beyond a simple GUI by providing comprehensive capabilities for process control, data collection, historical data analysis, alarm management, and remote access. Section 6 of the standard IEC 61512 - Batch control [62] describes functions that could be implemented for successful management and control of a batch process.

Some of these functions can be implemented as functions of SCADA.

For successful batch process execution, a system must know recipe parameters such as the amount of liquid for an add phase and temperature. For that reason, SCADA should be provided with input boxes. After starting the process it executes automatically phase by phase. However, operators must have the ability to respond to some situations like routine adjustments or emergency conditions. This is a task for manual control which is described in the standard as the ability to manipulate equipment entities directly by the operator.

Data collection is the next and important component within the domain of Process Control. This includes gathering and storing data from various sources, including sensors, derived values, and events. This historical record serves as a valuable resource for analysis. Stored data can be used for quality control and process improvement. Visualization of real-time and collected data can be implemented in a plot form which offers a dynamic and graphical representation of system parameters. It can help to quickly grasp trends and patterns aiding in informed decision-making.

Another thing that the standard describes is unpredictable events. Unpredictable events refer to incidents within a batch process that are not planned (alarms, equipment failures). These events are typically driven by irregular process conditions or physical factors. In the case of process alarms, the standard also offers the data related to these events (e.g., time of activation, time of acknowledgment, alarm limit). Figure 3.4 summarizes and shows the planned and previously described functionalities.

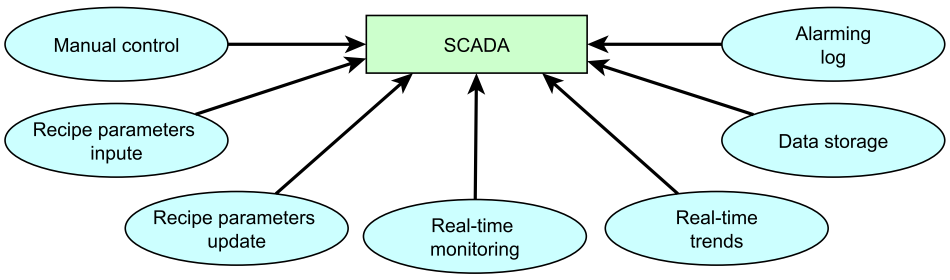


Figure 3.4: A planned functional of a SCADA. *Manual control* gives the possibility to turn on/off devices (e.g. mixer, pump) for test purposes or in case of an emergency situation. *Recipe parameters input* and *Recipe parameters update* are responsible for entering and updating information about process parameters. *Real-time monitoring* and *Real-time trends* provide visual information about the current state of the system in different forms. *Data storage* write data into a database for later usage and analysis. *Alarming log* displays all unpredictable events in the system.

CHAPTER 4

# Implementation of the concept for the batch process

This chapter describes the implementation of the concept of batch process control. It provides a detailed overview of the software used in this work and explains the principle of operation for each particular part of the project.

## 4.1 Hardware setup

To perform the process correctly, the proper connection, whether at the device or software level, is crucial. The laboratory setup consists of two connected to each other tank stations. Figure 4.1 depicts the Piping and Instrumentation diagram (P&ID).

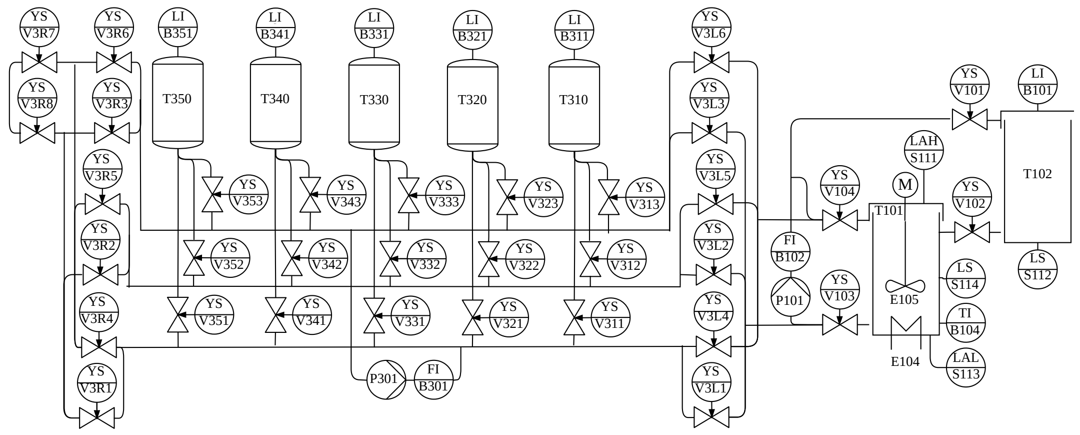


Figure 4.1: The Piping and Instrumentation diagram of the plant.

The first station comprises two tanks, T101 and T102. Tank T102 is equipped with one level sensor and one level switch. Tank T101 includes a heater, a mixer, and three level switches – two for indicating low and high liquid levels, and the last one to indicate if the liquid level is below the heater. The second station comprises five vessels, each equipped with level sensors. Additionally, each station is equipped with one pump, one turbine flowmeter, and several valves. To control the process a supervising laptop is used as the Master device and two EL1100 EtherCAT couplers as slave devices. Communication is performed via the EtherCAT protocol. Table 4.1 and Table 4.2 show how signals are split between the setups.

| Module | Signal |
|---|---|
| DO EL2008 | valves V101-V109, pump P101 |
| DO EL2008 | mixer E105, heater E104 |
| AO EL4004 | pump speed controller P101N |
| DI EL1018 | reed switch T101, float switch T102, cap. sensors B113, B114 |
| AI EL3702 | flow sensor B102, temperature sensor B104 |

Table 4.1: Signal table of the two tank Festo demonstrator setup.

| Module | Signal |
|---|---|
| DO EL2008 | pump P301, valves V3P1, V3L1-V3L6 |
| DO EL2008 | valves V311-V332 |
| DO EL2008 | valves V333-V353, V3R1 |
| DO EL2008 | valves V3R2-V3R8 |
| AO EL4004 | pump speed controller P301N |
| AI EL3064 | flow sensor B301, level sensors B311-B331 |
| AI 3174-0002 | level sensors B341, B351, B101 |

Table 4.2: Signal table of the five tank Festo demonstrator setup.

## 4.2 Skills implementation overview

For this project, 4diac IDE and 4diac FORTE are used to develop skills in the form of function blocks. It provides a development environment for creating control applications for industrial automation systems. To run these applications 4diac FORTE is used. 4diac IDE and 4diac FORTE are open-source software frameworks for industrial automation and control. They are based on the IEC 61499 standard for distributed automation systems. Both 4diac and FORTE are designed to be used in industrial automation systems, such as Programmable Logic Controllers (PLC) [84].

The skill-based approach focuses on breaking down complex tasks into smaller, simpler skills that can be easily implemented as function blocks. To maintain this approach and avoid increasing system complexity, only Simple Function Blocks (FB) are used for creating composite skills without using advanced features such as the Execution Control Chart (ECC) in 4diac IDE. This approach allows for easy reconfiguration, scalability, and robustness of the system. It also allows for the easy integration of new skills or modification of existing skills.

Not all of the necessary skills can be implemented as a function block because of the complexity of the skill task. Some of them are implemented as a service with the Python programming language. It does not mean that it is impossible to implement them as a function block, but this kind of implementation causes an unjustified increase in the system complexity which decreases in robustness and flexibility and significantly reduces the potential for reconfiguration.

Figure 4.2 shows how skills are divided inside the project. Each column indicates the project area to which a specific skill belongs. Blue skills are skills executed in a Python IDE, yellow in 4diac IDE, and orange in SCADA.

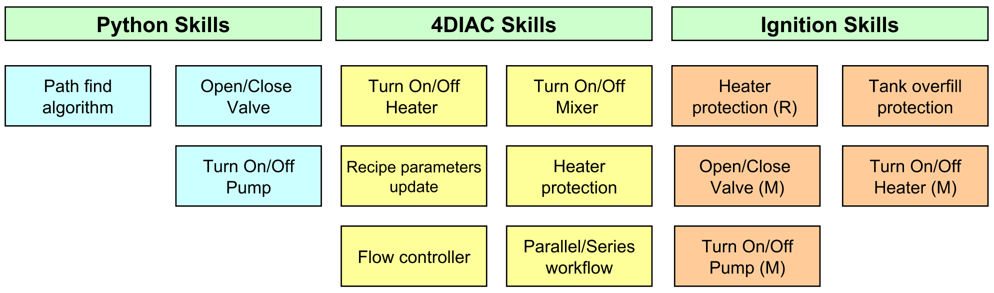


Figure 4.2: Separation of the skills within a project. (M) stands for manual control, (R) is for reserve protection measures.

## 4.3 Shortest path algorithm skill

The laboratory setup consists of a complex network of pipes and valves. In order to perform any action, such as an "Add" action, the system must be able to build a path between point A and point B and determine which valves need to be opened to construct this path.

As an efficient solution for this type of system, the representation of the redundant pipe system as a graph is chosen. Most valves allow fluid to flow in both directions, so the system must take this into account. The "Add" phase implies that the liquid must be transferred from one tank to another, so the path should contain not just a list of valves but also a pump for a successful execution of the process.

Figure 4.3 illustrates a directed graph of the entire system. Green nodes represent tanks, blue nodes represent pumps, and light blue nodes represent valves. Orange nodes represent pipe connections and play no role outside of the path planning algorithm.

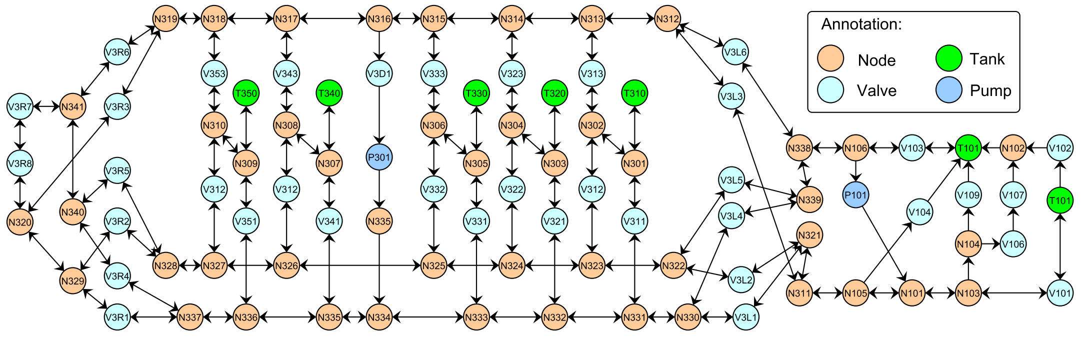


Figure 4.3: Representation of a redundant pipe system in a graph form

For the implementation of the directed graph in Python, the NetworkX package is used [85]. The graph is represented as a list of edges in the form:

```
edgeList=[['T101','V103',6],['V103','T101',6],['V104','T101',10],
                                        ...
          ['N335','N336',10],['N336','N335',10],['N336','N337',10]]
```

where the first term represents the initial node of the edge, the second is the target node of the edge, and the last one represents the weight of the edge. The length of the pipe is used as the weight of the edge. One set of arguments in brackets describes the edge in only one direction, so to build a bidirectional edge, a second set of brackets with the same weight but opposite order of nodes is required. Each node name must contain a specific letter (e.g. "V" stands for a valve, "T" - tank, "N" - node, and "P" - pump). This naming is strict and can not be changed for the successful execution of all designed skills in Python. From the list of supported by NetworkX algorithms, Dijkstra's algorithm is chosen as a search algorithm since it is the best and most efficient solution for weighted graphs with non-negative weights.

The pipe system has two pumps, but only one is needed for the "Add" phase execution. To define the best option for the path, the algorithm first looks for a sub-path and its weight from the source tank to one of the pumps, then the second sub-path from the pump to the target tank, as the one proposed in [12].

The algorithm then performs the same steps for the remaining pump and compares the weights. The path with the smallest weight is considered the shortest and is returned for further processing in the form of a list of nodes from the initial to the target tank.

## 4.4 Open/Close skills

These skills are responsible for the physical implementation of the path planning algorithm. Skills can open or close the path depending on the current task. Both skills use the list of nodes (an output of the shortest path algorithm) as an input parameter.

The basic principle of these skills is the same. After the execution of the shortest path algorithm, the skill receives the list of nodes. In the case of valves, skill search all nodes with the letter "V" in a node name, and subscribe to the directory where nodes of this type are placed. After that, the skill sends the command to change the value of these nodes according to the task (open or close). Same procedure but with "P" for pumps. The main difference between these skills is the order of execution. Figure 4.4 shows that in the case of the "Open" skill, it opens valves first and then the pump, whereas the "Close" skill does the same in reverse order to prevent any possible emergencies with the pressure (e.g. hydraulic shock) in closed pipes.

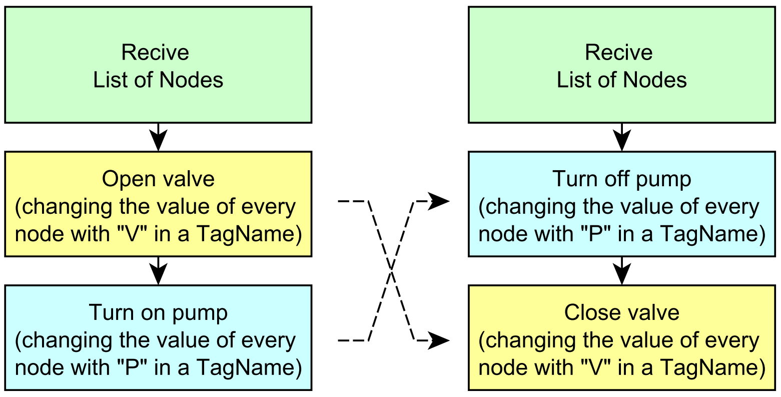

Recive
List of Nodes
Open valve
(changing the value of every node with "V" in a TagName)
Turn on pump
(changing the value of every node with "P" in a TagName)
Recive
List of Nodes
Turn off pump
(changing the value of every node with "P" in a TagName)
Close valve
(changing the value of every node with "V" in a TagName)

Figure 4.4: Execution order for Open and Close skills

## 4.5 Flow control skill

The flow control skill is the last part of the implementation of the add phase in the recipe. Two previous chapters explain how to find and build a path for a pumping process. The task that this skill performs is a volume calculation of a pumped liquid and task completion notification. It is a composite skill and is implemented as a composite Function Block (FB) type in 4diac IDE. As is shown in Figure 4.5 it contains several atomic skills. Each atomic skill has a simple (contains one algorithm) FB type.

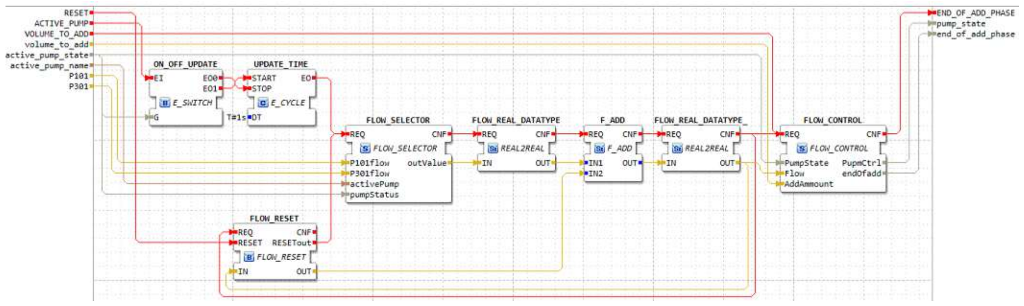


Figure 4.5: Internal structure of the flow control composite skill. Each FB is an atomic skill, provides encapsulated logic, and is implemented as a Simple FB in 4diac IDE.

When the shortest path is found, valves are opened, and a pump is turned on, the "Add" phase is ready to be executed. The triggering of the *active pump state* input indicates that one of the pumps is turned on and the "Add" phase is started. The *active pump* input contains the exact name of the turned-on pump. *P101* and *P301* receive the information from the flow meters. The **FLOW SELECTOR** function block decides which flow should be considered for a volume calculation. The **ADD** block calculates the amount of transferred liquid, and then the **FLOW CONTROL** compares the current and desired volumes. The outputs of the composite block *pump state* show if the pump is on or off and *endOfadd* indicates if the current task is finished. When the task is finished the system can call Close skill to turn off the pump and close the valves. Once the first "Add" phase is done, the *RESET* event resets the block. After that, the second "Add" phase can be started.

## 4.6 Heater composite skill

After the full execution of two add phases, when both liquids are in the desired tank, this mixture must be mixed and heated. Heater skill perform one of these tasks. Basically, for the successful execution of the heat phase, this skill should just check whether the current temperature in a tank matches the one in the task, and send a command to control the heater. However, the current implementation contains additional features. The idea behind this is to develop a core functionality, which can be improved with additional FBs. This should help to improve flexibility and help to fine-tune skills in a process. Figure 4.6 shows the internal structure and as with the previous FB, this one is also a composite FB, contains several atomic skills inside, and has the same FB types in 4diac IDE.

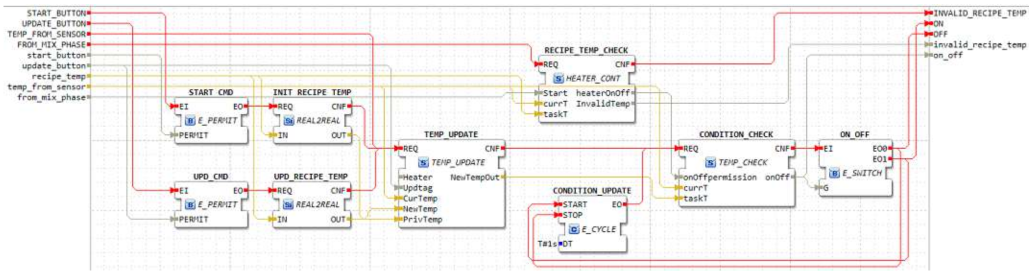


Figure 4.6: Internal structure of the Heater composite skill. Each FB is an atomic skill, provides encapsulated logic, and is implemented as a Simple FB in 4diac IDE.

The desired temperature comes from the *recipe temp* input by triggering the *start button* and *update button* inputs, respectively. The **TEMP UPDATE** function block decides if an update is possible, e.g. if the new temperature is equal to the current task temperature or less than the temperature in the tank, it ignores the update command. The **RECIPE TEMP CHECK** function block verifies if the initial task temperature is correct and sends an error message to the SCADA by using the *invalid recipe temp* output if it is not.

The **CONDITION CHECK** function block controls the heater state with the corresponding *on off* output. It receives the temperature data from the sensors and compares it with the desired temperature. Once the desired temperature is reached, the heater is turned off.

## 4.7 Mixer composite skill

According to the task, two liquids must be mixed to create a homogeneous solution. The mix phase is one of two phases that should be executed in parallel according to the recipe from the third chapter. The basic task is simple, this phase should be always active when the heat phase is active. Mixer composite skill performs mix phase and as the heater skill has extended functional to increase flexibility. Figure 4.7 shows the internal structure of the mixer skill.

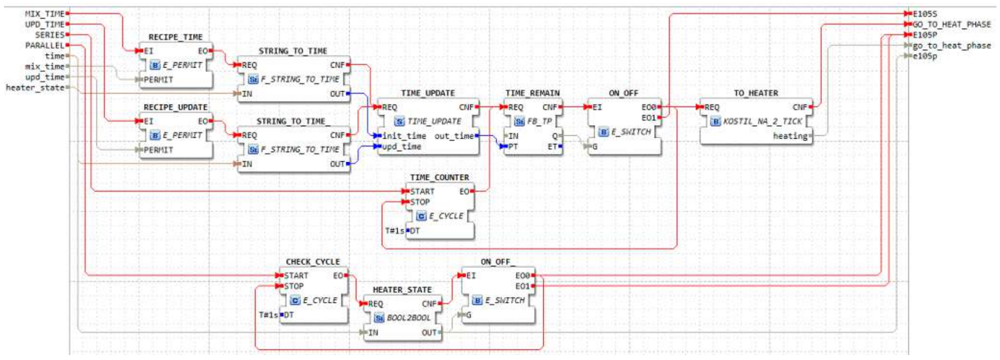


Figure 4.7: Internal structure of the Mixer composite skill. Each FB is an atomic skill, provides encapsulated logic, and is implemented as a Simple FB in 4diac IDE.

The current implementation can offer two working modes: series mode and parallel mode. The reason for that is to make this skill more useful outside of the current recipe, e.g. when liquid should be mixed during the exact amount of time to become desired consistency. The mode can be set through the corresponding input event.

The parallel mode (three lower FBs in Figure 4.7) keeps the mixer turned on if the heater is on. The **CHECK CYCLE** block checks every second the status of the heater. If the heater is on, the mixer is also turned on.

The series mode runs the mixer for a predefined time interval. In this project, these values can be set via SCADA. The **RECIPE TIME** and **RECIPE UP-DATE** function blocks send these time values (in seconds) through string-to-time data type converters to the **TIME UPDATE** function block. When the desired time is set, the mixer is turned on until the end of the timer. The time can be updated even during the active mixing phase. The *e105* output controls the mixer, and the *go to heat phase* output informs that the mixing is done and the process can further proceed.

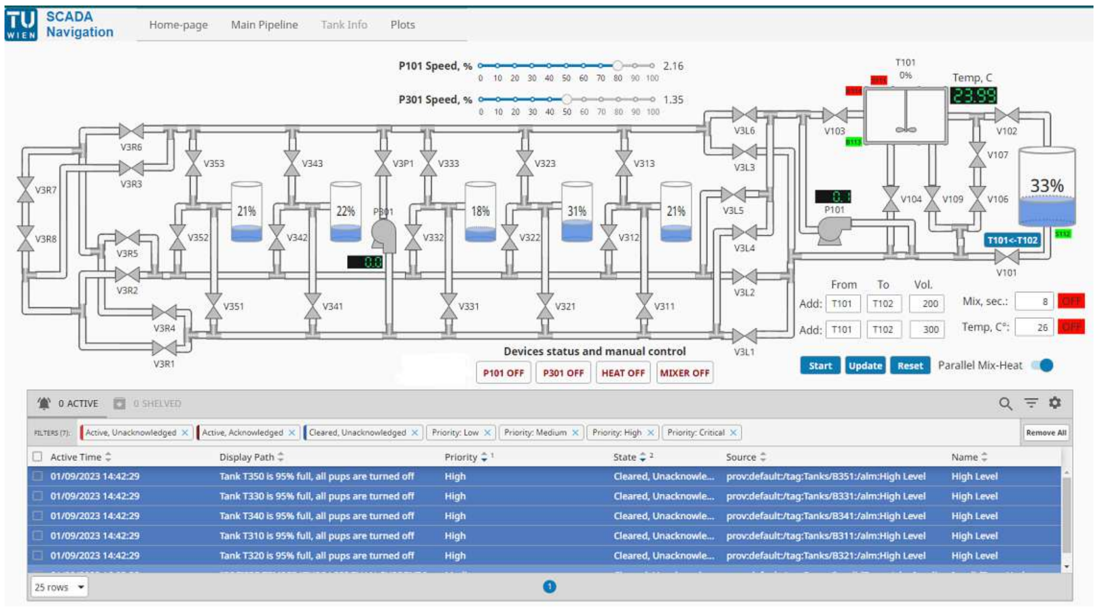
## 4.8 User interface

In order to control, observe, and modify the flow of the process, the proper user interface with sufficient functionality should be developed. Ignition SCADA by Inductive Automation is a great tool for developing this kind of interface.

Ignition SCADA by Inductive Automation is a software platform that is used for industrial automation and control systems. SCADA stands for Supervisory Control and Data Acquisition (SCADA). It is designed to provide real-time data collection, monitoring, and control of remote industrial assets, such as machines, pumps, valves, and sensors. Ignition SCADA software provides a variety of features, including a web-based interface, built-in drivers for connecting to various industrial protocols, alarming, reporting, and scripting capabilities. It can be used in a wide range of industries, including manufacturing, energy, water and wastewater, and transportation [86].

Figure 4.8 shows an implemented interface of the SCADA system. The schematic pipe system visualizes the flow of the process. It shows the status of valves and pumps with color indication, and displays values of the temperature sensor and flow meters. In addition, it shows the status of the reed switch and capacity sensors. Tank icons indicate the current level of the liquid.

For manual control, the SCADA can offer two sliders for pump speed control. Four buttons in the middle show the current status of the device (red - off, green - on) and provide manual control for devices. Three blue buttons are responsible for starting/updating the process and resetting all valves. The area on the right side serves for entering recipe-related information (e.g. temperature, amount of liquid for each phase).



| Active Time | Display Path | Priority | State | Source | Name |
|---|---|---|---|---|---|
| 01/09/2023 14:42:29 | Tank T350 is 95% full, all pups are turned off | High | Cleared, Unacknowle... | prov:default:/tag:Tanks/B351:/alm:High Level | High Level |
| 01/09/2023 14:42:29 | Tank T330 is 95% full, all pups are turned off | High | Cleared, Unacknowle... | prov:default:/tag:Tanks/B331:/alm:High Level | High Level |
| 01/09/2023 14:42:29 | Tank T340 is 95% full, all pups are turned off | High | Cleared, Unacknowle... | prov:default:/tag:Tanks/B341:/alm:High Level | High Level |
| 01/09/2023 14:42:29 | Tank T310 is 95% full, all pups are turned off | High | Cleared, Unacknowle... | prov:default:/tag:Tanks/B311:/alm:High Level | High Level |
| 01/09/2023 14:42:29 | Tank T320 is 95% full, all pups are turned off | High | Cleared, Unacknowle... | prov:default:/tag:Tanks/B321:/alm:High Level | High Level |

Figure 4.8: User interface of the SCADA system. Implemented SCADA can offer visualization, manual control, alarming, data input, real-time plotting capabilities, and data storage.

To provide data storage, an Ignition gateway is connected to an SQL lite database. It collects information about liquid levels, temperature, flow, and status of mixers, pumps, and heaters. The "Plots" tab on the top left corner provides access to real-time and history plots. All data can be exported in a CSV file. The alarming window below indicates such emergency events as tank overfilling, excessive draining of a tank, and liquid level below the heater. The alarm log contains information about time, priority, description of event, and status if this alarm was acknowledged by an operator or not.

It is also worth noting that UaExpert [87] third-party client was used to test the operation of the algorithm, intermediate values, and also as a main client to control the process before the above-described SCADA system was fully implemented (and as an additional and auxiliary after). It allows to explore the address space, read and write data, invoke methods, and monitor variables in OPC UA servers. Additionally, UaExpert provides features for testing and validating OPC UA implementations, which are discussed in the next section.

## 4.9 OPC UA communications

As previously mentioned, OPC UA is designed to provide a unified, secure, and reliable way for industrial devices to share information with each other and with higher-level systems such as SCADA systems. OPC UA uses a client-server model. Initially, the server of the Ignition gateway was considered the main candidate, however, all software and libraries in this work are free or open-source with varying levels of support. This can lead to compatibility problems. As a result, the open62541 [88] server integrated into FORTE shows fewer problems with communication than other candidates.

As described in Chapter 4.2 skills execution is divided between three different parts. Figure 4.9 shows the basic communication relationships between the specific modules of this project. The upper area represents an orchestrator (a laptop), and the lower part represents the physical plant. Data transfer between the plant and the laptop works via the EtherCAT protocol.

The TwinCAT module receives, converts (from analog 16-bit values to SI system units), updates data about the current state of the plant on the OPC UA server (e.g. temperature, flow, liquid level), and sends output control signals to the actuators.

The Ignition SCADA, 4DIAC, and Python blocks take the actual information they need via the OPC UA protocol right from the server and do not interact with each other. Nodes update their values continuously and regardless of the needs of other modules. Database SQL Lite block stores values and timestamps of tags for reports and plots.

Open62541 is an open-source implementation of the OPC UA communication protocol. It is written in the C programming language and is designed to be portable, memory-safe, and efficient. It can be used to create OPC UA clients and servers or to integrate OPC UA-based communication into existing software applications, and it is compatible with a wide range of operating systems and platforms [88]. In this project, FORTE is compiled with the open62541 source code.

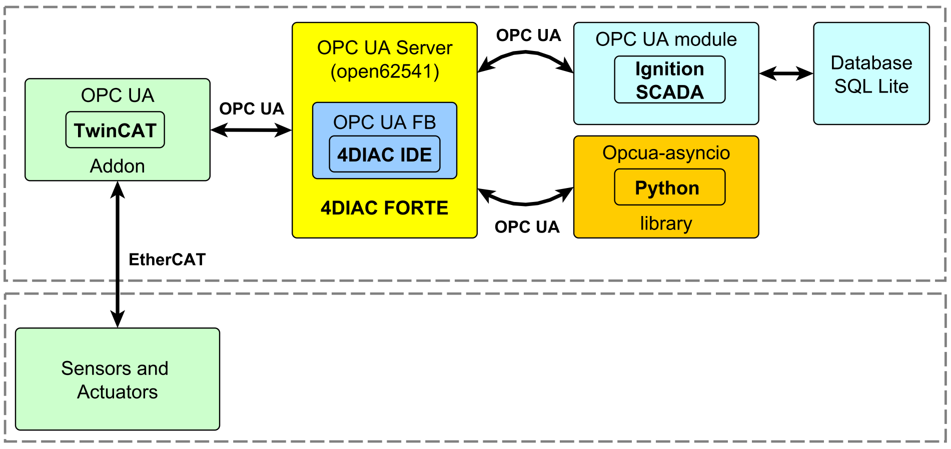


Figure 4.9: Basic scheme of communication between particular parts of the project. The upper part is proceeded on an orchestrator, lower part is a physical plant. Data transfer between the plant and orchestrator works via EtherCAT protocol. TwinCAT reads data from sensors, sends all these data to the OPC UA server (e.g. temperature, flow, liquid level), and sends output control signals to actuators. The SCADA, 4DIAC, and Python blocks take the actual information they need right from the server and do not interact with each other. Database SQL Lite contains values and timestamps of tags for the reports and plots.

During this process, two main settings should be taken into account: an endpoint and an update rate. The endpoint is basically a port on the computer that the OPC UA server will use (e.g. opc.tcp://localhost:4842/). The update rate is the frequency of data updates on the server and for this project, this value is equal to 200ms. To add nodes on the server 4diac IDE is used. One full functional node requires four FBs: SUBSCRIBE, PUBLISH, and two converters (e.g. INT2INT) FBs. The SUBSCRIBE and PUBLISH FBs contain specific information about the node such as name, place on the server, and type. Converters FBs tell which datatype the created variables should have (e.g. integer in case of INT2INT). Figure 4.10 shows the example of implementation for four boolean nodes.

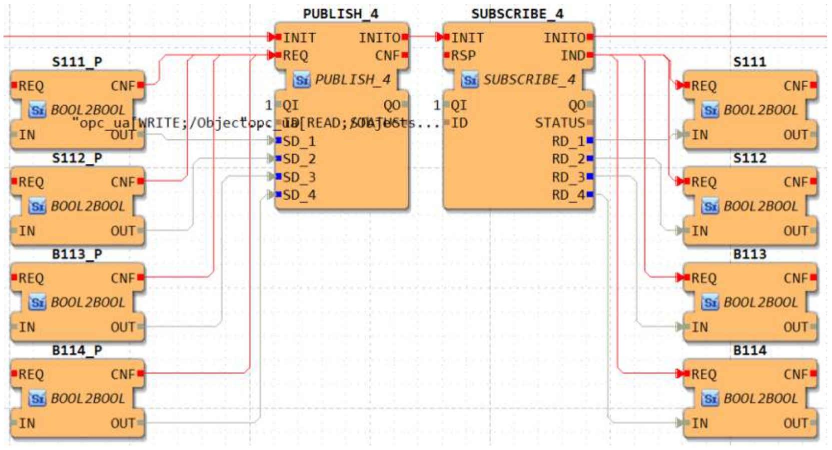
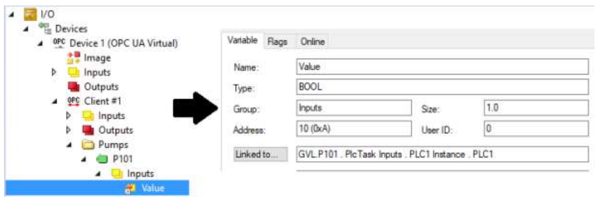


Figure 4.10: The picture displays the implementation of four boolean nodes in 4diac IDE. In this example, the SUBSCRIBE and PUBLISH FBs each have four outputs/inputs, which reduces the total number of FBs in the project. Consequently, each FB must include information about all four nodes. The BOOL2BOOL converters indicate that the data type used is boolean.

By default, TwinCAT does not support OPC UA functionality. To establish the connection, the additional software package TF6100-OPC-UA from the official website needs to be installed [89]. This addon allows to installation of a virtual OPC UA device to send and receive data. When the endpoint is set, all required tags should be connected manually to a special OPC UA variable in TwinCAT as shown in Figure 4.11. Then this variable can be connected to any other normal variable in TwinCAT.

Ignition can connect to third-party OPC servers via OPC UA by using the OPC COM module. This built-in module tool provides both server and client functionality and opens up the possibility of connecting Ignition to any device with servers [90]. Basically, the connection procedure is the same as in TwinCAT but Ignition provides a tool to automate this process and save time, e.g. UDT tags.



Name: Value
Type: BOOL
Group: Inputs Size: 1.0
Address: 10 (0xA) User ID: 0
Linked to... GVL.P101 . PlcTask Inputs . PLC1 Instance . PLC1

Figure 4.11: Example of nodes connection in TwinCAT. The figure shows a connected OPC UA virtual device, tag for the pump P101. It is shown that the input value of this tag is connected to the GVL.P101 OPC UA variable.

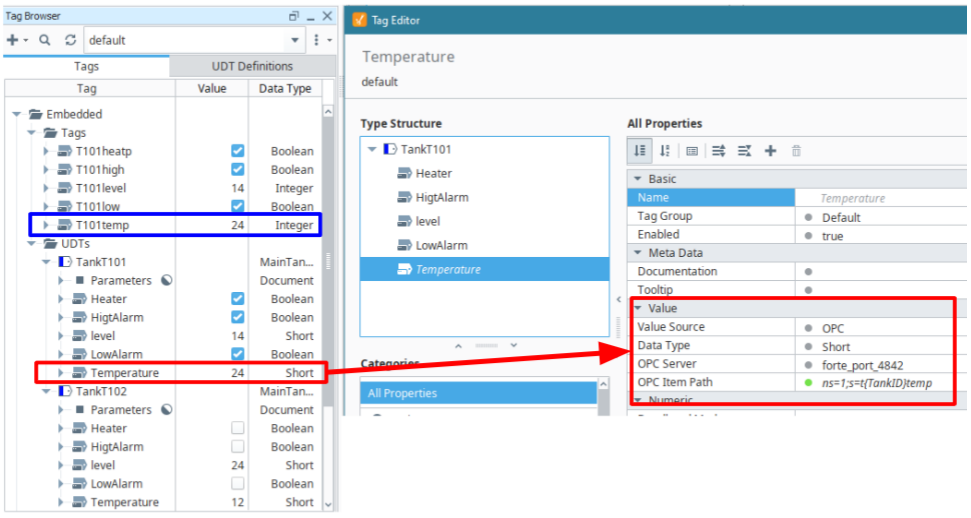
UDT stands for User Defined Type. A UDT is a feature that allows users to define custom data types in Ignition's Tag system. It enables the grouping of related parameters into a single user-defined type, making it easier to manage and organize complex data structures. Figure 4.12 shows the example of the implementation of UDT for a tank. This UDT contains such parameters (with different data types) as temperature, level, heater state, and high and low alarms. Each UDT has a changeable parameter (e.g. *TankID* in this particular case) which is used for the automatic creation of UDT instances. This is very similar to the data type mapping that is described in Chapter 2.1.2. Once UDT is created it can be automatically generated and connected to data on the OPC UA server for each equipment of the same type. UDTs can contain not just data from different servers but also information about scripts which are discussed in Chapter 4.10.

Python module uses the Opcua-asyncio library. It is an open-source, asynchronous OPC UA client and server based on the IEC 62541 standard. The library supports common OPC UA features such as data types, security, and subscriptions [91]. The connection procedure here is similar to the procedure in 4diac. The library requires all specific information about every node such as name, place on the server, type, etc.

## 4.10 Safety measurements implementation

Modern automation systems should be able to prevent accidents during the process execution to ensure reliability and safety of the plant. To keep a high level of scalability and to avoid increasing system complexity, most of the safety measurements are implemented in Ignition.

For advanced implementation, Ignition offers a scripting feature [92]. It allows to creation of a script (a service according to SOA) that runs on an OPC UA tag change. The scripting provides functionality that executes every time a node changes its value, even if the SCADA application is closed (but the Gateway, Ignition's core that proceeds data, must stay active). This is useful for implementing safety



| Tag | Value | Data Type |
|---|---|---|
| Embedded | | |
| Tags | | |
| T101heatp | ✓ | Boolean |
| T101high | ✓ | Boolean |
| T101level | 14 | Integer |
| T101low | ✓ | Boolean |
| T101temp | 24 | Integer |
| UDTs | | |
| TankT101 | | MainTan... |
| Parameters | | Document |
| Heater | ✓ | Boolean |
| HigtAlarm | ✓ | Boolean |
| level | 14 | Short |
| LowAlarm | ✓ | Boolean |
| Temperature | 24 | Short |
| TankT102 | | MainTan... |
| Parameters | | Document |
| Heater | ☐ | Boolean |
| HigtAlarm | ☐ | Boolean |
| level | 24 | Short |
| LowAlarm | ☐ | Boolean |
| Temperature | 12 | Short |

Temperature
default

TankT101
- Heater
- HigtAlarm
- level
- LowAlarm
- *Temperature*

| | |
|---|---|
| Basic | |
| Name | *Temperature* |
| Tag Group | Default |
| Enabled | true |
| Meta Data | |
| Documentation | |
| Tooltip | |
| Value | |
| Value Source | OPC |
| Data Type | Short |
| OPC Server | forte_port_4842 |
| OPC Item Path | *ns=1;s=t{TankID}temp* |
| Numeric | |

Figure 4.12: Example of a UDT tag. It can be observed that the parameter Temperature in the UDT description contains all the information (red right box) for the proper data mapping, e.g. port of the OPC UA server, namespace, and tag name in form t(*TankID*)temp. For example, the temperature tag for tank 101 (blue box) has a value of 24. As a result, this value is linked to the parameter Temperature (red left box) for UDT with *TankID* - 101.

measures that are always active, even when the SCADA is not being used. Python language is used for script implementation. An example of the script for tank overfilling event is shown below:

```
def valueChanged(tag,tagPath,previousValue,currentValue,
        initialChange,missedEvents):
        currentVolume = currentValue.value
        paths = ["[default]/Errors/capacityOftankID.value"]
        tankVolumeOverall = system.tag.readBlocking(paths)
        tankVolumeCrit = 0.95 * tankVolumeOverall[0].value
        if currentVolume >= tankVolumeCrit:
                system.tag.writeBlocking(["[default]/Pumps/P101",
                    "[default]/Pumps/P301"], [0,0])
```

The script turns the pumps off when the value of the tag that shows the current volume (*currentVolume*) in a tank shows 95 percent of the overall tank volume (*tankVolumeOverall*). This script is implemented for each tag of the corresponding tank by changing *capacityOftankID*. Measurements for preventing excessive draining of tanks are also implemented with scripting.

Another approach for the implementation of safety measurements is to use both function blocks and scripting tools. Heater protection skill prevents the heater from turning on if the liquid level is lower than the heater and protects a relay from multiple triggering. A temperature sensor noise can cause multiple triggering of the relay when the temperature value is a condition for a heater's work. Implementation of this composite skill in 4diac IDE is shown in Figure 4.13.

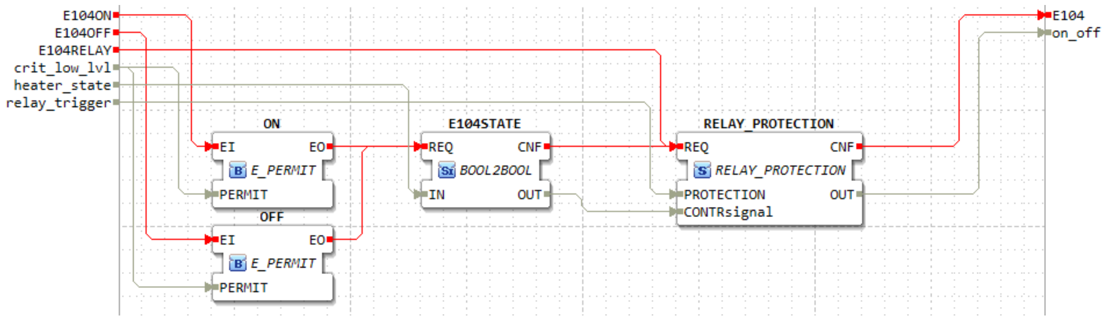


Figure 4.13: Composite skill for the heater and relay protection

This skill receives the heater level switch state (*crit_low_lvl*) and information if the heater was activated before (*relay_trigger*). SCADA latch a tag if the previous state of the heater is ON and it changes to OFF. This tag unlatches itself when a new recipe begins. If at least one of the conditions is not met, the skill does not pass the control signal further. This approach shows how different tools can be combined. It can help combine the strengths of each approach and create more complex solutions for some tasks. Overall, the use of both scripting and function blocks in 4diac IDE allows for flexible and efficient implementation of safety measurements in the batch process control system.

# CHAPTER 5

# Evaluation of results

This chapter describes the implementation of a skill-based approach for batch process control. It presents an evaluation of OPC UA implementation, and the path planning algorithm and provides an overview of the manual control possibility. Additionally, the functionality of each phase of the recipe is discussed.

## 5.1 Path planning algorithm

The path planning algorithm shows proper work as expected. Figure 5.1 shows an execution log of the Python shortest path algorithm. The algorithm calculates the path and weight for each pump. Then, it compares the weights and chooses as the output the optimal list of nodes (the one with the smallest weight). In this particular case, the path between tanks T340 and T101 through pump P301 may appear longer (18 nodes for P301 versus 16 nodes for P101). However, this path has a slightly smaller weight (220 versus 227), which means that it is physically shorter, and that is why it is chosen as the optimal path. Furthermore, this list is used in such simple tasks as "Open valves" and "Close valves".

## 5.2 Manual pump control

Figure 5.2 shows the ability for manual flow control. The flow control occurs by changing the position of sliders, one for each pump, via the SCADA interface. The picture demonstrates full freedom during control. With the maximum output of the 16-bit analog signal (0-32767), the flow meter shows 45-46 ml/sec of flow, which is equal to 2.7 L/min. After turning off the pump, an exponential decay is observed.

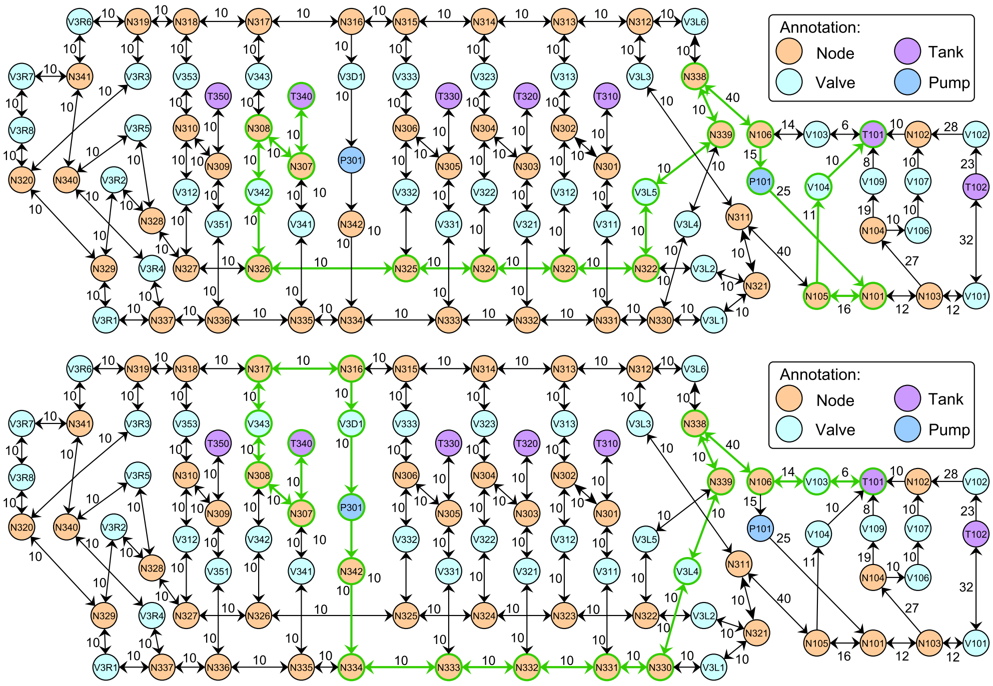


Figure 5.1: The result of the shortest path algorithm. The top picture shows the final path through the pump P101. This path contains 16 nodes and has a weight of 227. The bottom picture is for the pump P301. It has 18 nodes and has a weight of 220. This means that even though the second path appears longer because of the greater number of nodes, it is actually physically shorter and, therefore, the optimal choice.

One of the possible reasons for this is flowmeter characteristics. Turbine flowmeters have different characteristics and response times. Another reason is pressure distribution. When the pump is turned off, the pressure in the system starts to decrease. The relationship between the fluid flow rate and pressure can be nonlinear. Residual flow reflected from a closed valve leaf can cause slight backflow. This backflow can influence the flowmeter readings after the pump is turned off. Each factor separately or their combination could cause an exponential decay.

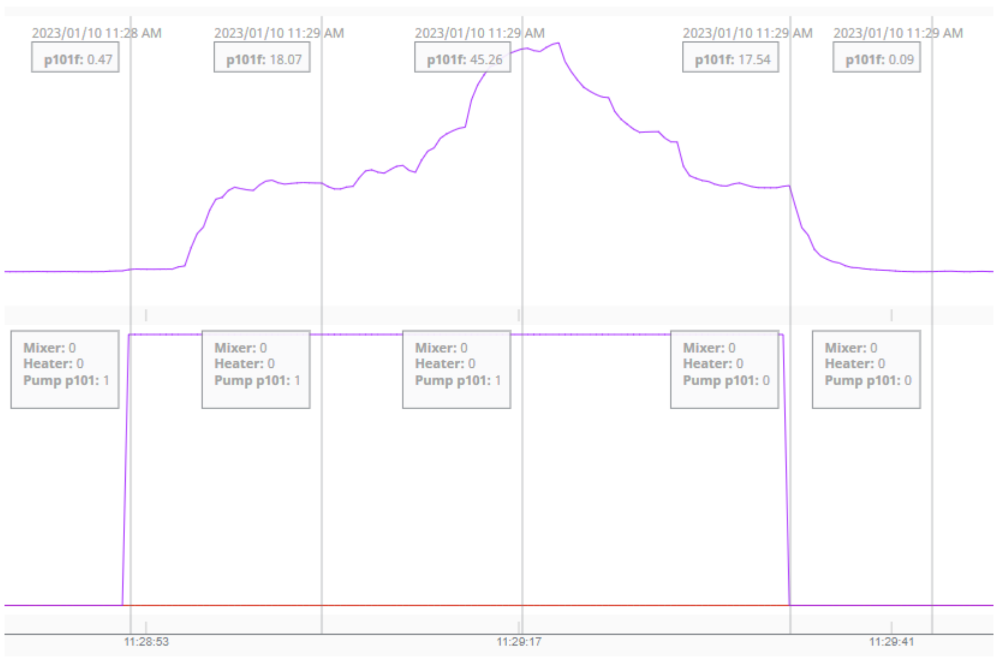


Figure 5.2: Manual pump control. The upper plot shows pumping speed control, the lower plot shows the on/off signal. The plot demonstrates the possibility of free control. On maximum pumping power, the pump achieves 45-46 ml/sec. After turning off the pump, an exponential decay of the flow is observed.

## 5.3 Execution of the "Add" phase

After the shortest path is calculated for a particular part of the "Add" process, it is ready to be used for further processing. The list of nodes contains the names of all valves for path construction and the name of the pump that should be opened for the "Add" phase execution.

Figure 5.3 illustrates the execution of two "Add" phases. The task is to add 150 ml of a liquid to tank T102 during the first phase and then add an additional 300 ml. The result of the task is equal to 156 ml and 304 ml, respectively. The delay between "Add" phases is artificial and set for the purpose of differentiating the two processes for clear and simple analysis. This pause duration can be modified manually as needed.

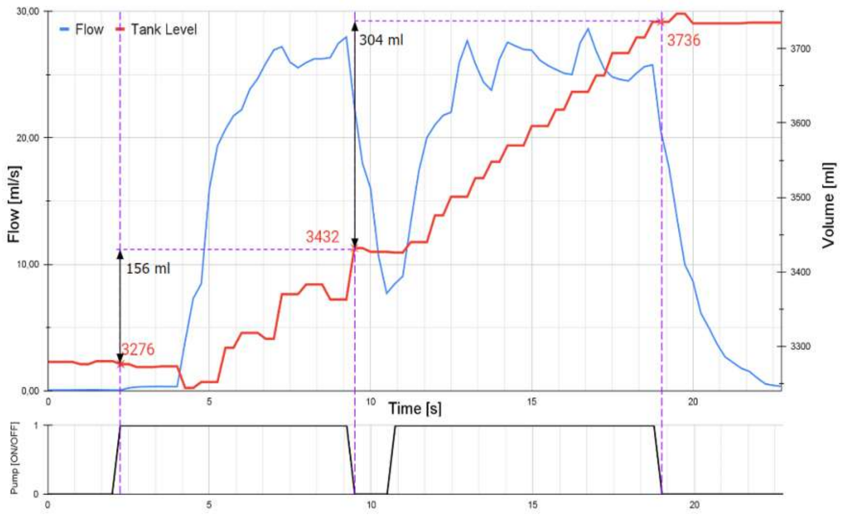


Figure 5.3: Execution of "Add" phases. The task is to add 150 ml of a liquid to tank T102 during the first phase and 300 ml during the second phase. On the top plot, the red line shows the tank level and the blue shows the flow. The bottom image shows the pump state. It can be observed that the final amount of pumped liquid is equal to 156 ml for the first phase, and 304 ml for the second.

## 5.4 Temperature control and parallel workflow

As soon as "Add" phase is complete the next step is the "Mix" phase in series mode or "Mix" and "Heat" in case of parallel mode. Figure 5.4 demonstrates a parallel execution of "Mix" and "Heat" phases with the task of heating a liquid up to 25°C degrees. The picture shows that the heater is turned off at a temperature of 24.8°C degrees. This can be explained by the write rate of the database. The temperature sensor value changes quickly within a small range, and the number of these values is much higher than the number of values the database can save in a time interval, causing the database to skip some values. However, the heat controller does not have this issue, which means that at this timestamp, the function block received the desired value of 25°C degrees and turned off the heater.

Figure 5.4 illustrates that the temperature continues to increase up to an additional 1°C degree even though the heater is turned off. This effect is caused by residual heat retained in the heater after it is turned off.

It is also noticeable that there is a small time difference between the turning on/off of the heater and mixer. The duration of this delay is 1 second, which corresponds to the update rate of the mixer function block. The condition for the mixer during parallel mode is the current status of the heater. In simple terms, the mixer is active only when the heater is active.

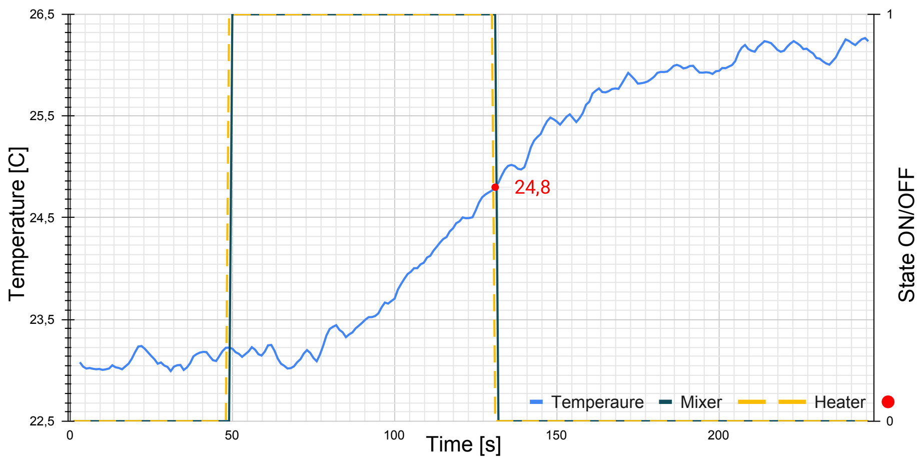


Figure 5.4: Parallel execution of "Heat" and "Mix" phases. The lower plot shows the current status of actuators. Residual heating of the heater. The figure shows that in spite of the task to heat a liquid up to 25°C, the temperature keeps increasing around up to 26.3°C. This effect causes a residual heat of the heating element.

CHAPTER 6

# Conclusion and Further work

This thesis demonstrates the implementation of a recipe-based batch process using a combination of Service-Oriented Architecture and a Skill Engineering Model. This approach implies implementing a process as a sequence of independent entities with encapsulated logic skills. In a similar way, the IEC 61512 Batch control standard describes the design of a batch process. This makes refining a complex task into a sequence of simple actions a common feature, making this approach a suitable candidate for implementation. The project itself was successfully tested on a laboratory setup.

Research question 1: *What kind of advantage/disadvantage has detailed refining of an action into a sequence of atomic skills over a more common skill description*?

The level of refining for a skill describes the flexibility and reusability that this skill offers to a system. Atomic skills with simple functions can be easily reused across different composite skills, e.g. cycles (E_CYCLE FB), and switches (E_SWITCH FB). This reduces efforts for the development of the same functionality in comparison to a more common skill description. In case a process evolves under new requirements, the process stage can be extended or modified with existing atomic skills without affecting the entire process as shown in heat phase implementation. It was also noticed during development that in the case of detailed refining is much easier to identify and fix problems in an algorithm. However, the usage of skills with primitive functions seems excessive and contradicts to coarse-grained attribute of a service in SOA. In theory, an excessive refinement can also lead to increased complexity of the system due to the need to manage and coordinate multiple atomic skills for a single action, but the current project is not huge enough to verify this theory.

Research question 2: *Does this approach potentially able to increase the flexibility of the plant*?

The answer is definitely positive. The module system allows specific process modifications without affecting other parts of the system because each implemented module is independent. A similar situation with skills and services. Each of them encapsulates its own functionality which means that changes in a particular skill or a service have no effect on the rest of the system. 4DIAC IDE with its runtime environment FORTE ensures execution of algorithm even without physical PLC which is the case of the current work and allows to use of skills implemented by different developers without compatibility problems.

Using OPC UA as middleware software introduces this project an element of Plug and Produce functionality to the system but under the assumption of proper naming of tags and strict data structure on the OPC UA server. OPC UA provides communication and data exchange to different software components like 4DIAC IDE, Ignition, TniwCAT, and Python. Its availability across different programming languages (e.g. Python, C++, Java), different PLCs vendors (e.g. Beckhoff, ABB, Siemens, etc.), and operational systems increase the number of possible tools for the implementation of additional modules. The example of UaExpert client usage shows that a new component can be simply connected to the server at any time. Updating an old version of a module can be accomplished within seconds simply by connecting a new version of the module to the OPC UA server.

Ignition offers features for fast reconfiguration of a user interface, e.g. UDT in combination with template views. This can be very useful when a large number of similar elements exist in a system. For example, only one UDT and template view should be implemented for the implementation of visual components for 39 valves in the system.

As the next step for further project implementation, the option to modify the graph online is a viable choice. In the case of an emergency, this capability can assist in isolating the damaged area of the pipe system or a single faulty valve, preventing the shutdown of the entire plant. Implementing the ability to execute two or more parallel recipes will significantly enhance the plant's efficiency.

Despite the fact that 4DIAC can establish a robust connection via OPC UA, the current state of OPC UA functionality is limited. This limitation can be problematic in situations where system updates/changes are necessary and plant downtime is unwelcome. The current implementation requires stopping the 4DIAC module first and then reloading it to apply changes. Additionally, the fact that a single OPC UA tag requires four function blocks, even if it is not used by the 4DIAC module, results in a massive and poorly adaptable OPC tag structure. Implementing this functionality as services for a Python module could offer a solution to this problem.

However, even the current state of the implementation shows a strong potential for the development of flexible, scalable systems for batch processes based on the combination of the Skill-based Engineering Model with Service-Oriented Architecture.

# Bibliography

[1] G. Marzano and A. Martinovs, “TEACHING INDUSTRY 4.0,” *SOCIETY. INTEGRATION. EDUCATION. Proceedings of the International Scientific Conference*, vol. 2, p. 69, May 20, 2020.

[2] O. A. Shvetsova and A. D. Kuzmina, “Development of engineering personnel in the era of the fourth industrial revolution,” in *2018 Third International Conference on Human Factors in Complex Technical Systems and Environments (ERGO)s and Environments (ERGO)*, St. Petersburg: IEEE, Jul. 2018, pp. 45–48.

[3] F. Sherwani, M. M. Asad, and B. Ibrahim, “Collaborative robots and industrial revolution 4.0 (IR 4.0),” in *2020 International Conference on Emerging Trends in Smart Technologies (ICETST)*, Karachi, Pakistan: IEEE, Mar. 2020, pp. 1–5.

[4] H. Hirsch-Kreinsen, “Industrie 4.0,” in *Handbuch Innovationsforschung*, B. Blättel-Mink, I. Schulz-Schaeffer, and A. Windeler, Eds., Wiesbaden: Springer Fachmedien Wiesbaden, 2020, pp. 1–16.

[5] E. Gizem, “How to define industry 4.0: The main pillars of industry 4.0,” Nov. 2017.

[6] Y. Lu, “The current status and developing trends of industry 4.0: A review,” *Information Systems Frontiers*, Nov. 9, 2021.

[7] EPICOR, *What is industry 4.0—the industrial internet of things (iiot)?* Oct. 2020.

[8] U. D. Atmojo, Z. Salcic, K. I.-K. Wang, and V. Vyatkin, “A service-oriented programming approach for dynamic distributed manufacturing systems,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 151–160, Jan. 2020.

[9] F. Herrmann, "The smart factory and its risks," *Systems*, vol. 6, no. 4, p. 38, Oct. 26, 2018.

[10] L. Bassi, "Industry 4.0: Hope, hype or revolution?" In *2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)*, Modena, Italy: IEEE, Sep. 2017, pp. 1–6.

[11] S. Malakuti, J. Bock, M. Weser, P. Venet, P. Zimmermann, M. Wiegand, J. Grothoff, C. Wagner, and A. Bayha, "Challenges in skill-based engineering of industrial automation systems," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin, Italy: IEEE, Sep. 2018, pp. 67–74.

[12] M. Melik Merkumians, M. Baierling, and G. Schitter, "A service-oriented domain specific language programming approach for batch processes," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Berlin, Germany: IEEE, Sep. 2016, pp. 1–9.

[13] K. Holley and A. Arsanjani, *100 SOA questions: asked and answered*. Upper Saddle River, NJ: Prentice Hall, 2011, 242 pp.

[14] N. M. Josuttis, *SOA in practice*, 1st ed. Beijing ; Sebastopol: O'Reilly, 2007, 324 pp.

[15] T. Erl, *Service-oriented architecture: concepts, technology, and design*, 9. print. Upper Saddle River, NJ Munich: Prentice Hall PTR, 2009, 760 pp.

[16] A. Arsanjani, *Service-oriented modeling and architecture*, 2004.

[17] IETF, *Hypertext transfer protocol (http/1.1): Semantics and content*, 2014.

[18] K. A. Sedek, M. A. Omar, and S. Sulaiman, "Interoperable SOA-based architecture for e-government portal," in *2012 IEEE Conference on Open Systems*, Kuala Lumpur, Malaysia: IEEE, Oct. 2012, pp. 1–6.

[19] N. Kaur, R. Harrison, and A. A. West, "A service-oriented approach to embedded component-based manufacturing automation," in *2015 IEEE International Conference on Industrial Technology (ICIT)*, Seville: IEEE, Mar. 2015, pp. 2964–2969.

[20] R. Harrison, R. P. Monfared, and L. Lee, "Business driven engineering for powertrain industry," in *2009 IEEE Conference on Emerging Technologies & Factory Automation*, Palma de Mallorca, Spain: IEEE, Sep. 2009, pp. 1–4.

[21] M. Ghallab, D. S. Nau, and P. Traverso, *Automated planning and acting*. New York, NY: Cambridge University Press, 2016.

[22] D. L. Poole and A. K. Mackworth, *Artificial intelligence: foundations of computational agents*, second edition. Cambridge New York, NY Port Melbourne Delhi Singapore: Cambridge University Press, 2018, 792 pp.

[23] K. Dorofeev and A. Zoitl, “Skill-based engineering approach using OPC UA programs,” in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, Porto: IEEE, Jul. 2018, pp. 1098–1103.

[24] K. Dorofeev, C.-H. Cheng, M. Guedes, P. Ferreira, S. Profanter, and A. Zoitl, “Device adapter concept towards enabling plug&produce production environments,” in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Limassol: IEEE, Sep. 2017, pp. 1–8.

[25] M. Onori, N. Lohse, J. Barata, and C. Hanisch, “The IDEAS project: Plug & produce at shop-floor level,” *Assembly Automation*, vol. 32, no. 2, pp. 124–134, Apr. 6, 2012.

[26] S. Cavin, P. Ferreira, and N. Lohse, “Dynamic skill allocation methodology for evolvable assembly systems,” in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, Bochum, Germany: IEEE, Jul. 2013, pp. 218–223.

[27] S. Cavin and N. Lohse, “Multi-level skill-based allocation methodology for evolvable assembly systems,” in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, Porto Alegre RS, Brazil: IEEE, Jul. 2014, pp. 532–537.

[28] J. Ferreira, L. Ribeiro, P. Neves, H. Akillioglu, M. Onori, and J. Barata, “Visualization tool to support multi-agent mechatronic based systems,” in *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, Montreal, QC, Canada: IEEE, Oct. 2012, pp. 4372–4377.

[29] K. Evers, J. R. Seyler, V. Aravantinos, L. Lucio, and A. Mehdi, “Roadmap to skill based systems engineering,” in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain: IEEE, Sep. 2019, pp. 1093–1100.

[30] A. Bayha, L. Lúcio, V. Aravantinos, K. Miyamoto, and G. Igna, “Factory product lines: Tackling the compatibility problem,” in *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, Salvador Brazil: ACM, Jan. 27, 2016, pp. 57–64.

[31] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert, and A. Knoll, “Intuitive instruction of industrial robots: Semantic process descriptions for small lot production,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea: IEEE, Oct. 2016, pp. 2293–2300.

[32] A. Perzylo, N. Somani, M. Rickert, and A. Knoll, "An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany: IEEE, Sep. 2015, pp. 4197–4203.

[33] "IEEE standard ontologies for robotics and automation," IEEE, ISBN: 9780738196503.

[34] P. Jhunjhunwala, U. D. Atmojo, and V. Vyatkin, "Applying skill-based engineering using OPC-UA in production system with a digital twin," in *2021 IEEE 30th International Symposium on Industrial Electronics (ISIE)*, Kyoto, Japan: IEEE, Jun. 20, 2021, pp. 1–6.

[35] P. Zimmermann, E. Axmann, B. Brandenbourger, K. Dorofeev, A. Mankowski, and P. Zanini, "Skill-based engineering and control on field-device-level with OPC UA," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain: IEEE, Sep. 2019, pp. 1101–1108.

[36] K. Dorofeev, "Skill-based engineering in industrial automation domain: Skills modeling and orchestration," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*, Seoul South Korea: ACM, Jun. 27, 2020, pp. 158–161.

[37] N. Keddis, G. Kainz, and A. Zoitl, "Capability-based planning and scheduling for adaptable manufacturing systems," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, Barcelona, Spain: IEEE, Sep. 2014, pp. 1–8.

[38] K. Aleksandrov, V. Schubert, and J. Ovtcharova, "Skill-based asset management: A PLM-approach for reconfigurable production systems," in *Product Lifecycle Management for a Global Market*, S. Fukuda, A. Bernard, B. Gurumoorthy, and A. Bouras, Eds., vol. 442, Series Title: IFIP Advances in Information and Communication Technology, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 465–474.

[39] A. Kocher, C. Hildebrandt, L. M. Vieira Da Silva, and A. Fay, "A formal capability and skill model for use in plug and produce scenarios," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Vienna, Austria: IEEE, Sep. 2020, pp. 1663–1670.

[40] U. D. Atmojo, J. O. Blech, and V. Vyatkin, "A plug and produce-inspired approach in distributed control architecture: A flexible assembly line and product centric control example," in *2020 IEEE International Conference*

*on Industrial Technology (ICIT)*, Buenos Aires, Argentina: IEEE, Feb. 2020, pp. 271–277.

[41] A. Rocha, G. Di Orio, J. Barata, N. Antzoulatos, E. Castro, D. Scrimieri, S. Ratchev, and L. Ribeiro, "An agent based framework to support plug and produce," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, Porto Alegre RS, Brazil: IEEE, Jul. 2014, pp. 504–510.

[42] T. Arai, Y. Aiyama, Y. Maeda, M. Sugi, and J. Ota, "Agile assembly system by "plug and produce"," *CIRP Annals*, vol. 49, no. 1, pp. 1–4, 2000.

[43] P. Ferreira and N. Lohse, "Configuration model for evolvable assembly systems," in *4th CIRP Conference On Assembly Technologies And Systems*, May 2012.

[44] P. Danny, P. Ferreira, N. Lohse, and M. Guedes, "An AutomationML model for plug-and-produce assembly systems," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, Emden: IEEE, Jul. 2017, pp. 849–854.

[45] *Enas-energieautarke aktoren und sensoren.* `https://www.energieautark.com`, Sep. 2019.

[46] S. K. Panda, T. Schroder, L. Wisniewski, and C. Diedrich, "Plug&produce integration of components into OPC UA based data-space," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin: IEEE, Sep. 2018, pp. 1095–1100.

[47] OPC Foundation, *OPC 10000-5: UA Part 5: Information Model. B.4.5 FiniteStateMachineType*, `https://reference.opcfoundation.org/Core/Part5/v104/docs/B.4.5`, Oct. 2023.

[48] ——, *OPC 10000-10: OPC Unified Architecture. Part 10: Programs. Release 1.04.* `https://reference.opcfoundation.org/v104/Core/docs/Part10/4`, Nov. 2017.

[49] P. Danny, P. Ferreira, N. Lohse, and K. Dorofeev, "An event-based AutomationML model for the process execution of plug-and-produce' assembly systems," in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*, Porto: IEEE, Jul. 2018, pp. 49–54.

[50] Loughborough University, *openMOS project. Deliverable: D3.1. Open Plug and Produce Architecture Specification.* `https://www.openmos.eu/downloads/deliverables/`, Dec. 2020.

[51] M. Schleipen, R. Henßen, M. Damm, A. Lüder, N. Schmidt, O. Sauer, and S. Hoppe, "Opc ua and automationml - collaboration partners for one common goal: Industry 4.0," in *AutomationML user conference*, Sep. 2014.

[52] R. Henßen and M. Schleipen, "Interoperability between OPC UA and AutomationML," *Procedia CIRP*, vol. 25, pp. 297–304, 2014.

[53] X. Ye and S. H. Hong, "An AutomationML/OPC UA-based industry 4.0 solution for a manufacturing system," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, Turin: IEEE, Sep. 2018, pp. 543–550.

[54] P. E. Black and P. J. Tanenbaum, *"Graph", in dictionary of algorithms and data structures [online], last access 08 oct 2023*, `https://www.nist.gov/dads/HTML/graph.html`, Nov. 2020.

[55] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, Fourth edition. Cambridge, Massachusett: The MIT Press, 2022, 1291 pp.

[56] S. S. Skiena, *The algorithm design manual*, 2nd ed. London: Springer, 2008, 730 pp.

[57] R. Kempepatil and V. V. Rudraswamymath, "Advanced study of shortest route problem and its applications," in *International Journal of Scientific Development and Research (IJSDR)*, Jun. 2023.

[58] A. Madkour, W. G. Aref, F. U. Rehman, M. A. Rahman, and S. Basalamah, "A survey of shortest-path algorithms," 2017, Publisher: arXiv Version Number: 1.

[59] M. Okwu and I. Emovon, "Application of johnson's algorithm in processing jobs through two-machine system," *Journal of Mechanical and Energy Engineering*, vol. 4, no. 1, pp. 33–38, Aug. 9, 2020.

[60] D. Ivanova, I. Batchkova, S. Panjaitan, F. Wagner, and G. Frey, "Combining IEC 61499 and ISA s88 for batch control," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 187–192, 2009.

[61] C. Johnsson, "White paper s88 for beginners," in *World Batch Forum*, Jun. 2004.

[62] *Batch control. Part 1, Models and terminology.* Research Triangle Park, North Carolina: ISA, 1995.

[63] IEC:61512-1:1997, *Batch control - part 1: Models and terminology*, 1997.

[64] Siemens, *Engineering and automation of batch processes with pcs 7 along isa-88 models*, `https://cache.industry.siemens.com/dl/files/331/109784331/att_1052440/v2/109784331_Batch_processes_PCS_7_ISA_88_Docu_V1_en.pdf`, Jan. 2021.

[65] *Modeling and control of batch processes.* New York, NY: Springer Berlin Heidelberg, 2018.

[66] D. Ivanova, I. Batchkova, S. Panjaitan, F. Wagner, and G. Frey, "Combining IEC 61499 and ISA s88 for batch control," *IFAC Proceedings Volumes*, vol. 42, no. 4, pp. 187–192, 2009.

[67] A. Garcia, X. Oregui, U. Arrieta, and I. Valverde, "Methodology and tools to integrate industry 4.0 CPS into process design and management: ISA-88 use case," *Information*, vol. 13, no. 5, p. 226, Apr. 28, 2022.

[68] M. Vegetti and G. Henning, "Isa-88 formalization. a step towards its integration with the isa-95 standard," in *6th Workshop on Formal Ontologies meet Industry*, Feb. 2015.

[69] M. De Minicis, F. Giordano, F. Poli, and M. M. Schiraldi, "Recipe development process re-design with ANSI/ISA-88 batch control standard in the pharmaceutical industry," *International Journal of Engineering Business Management*, vol. 6, p. 16, Jan. 1, 2014.

[70] *Batch control. Part 2, Data structures and guidelines for languages*. Research Triangle Park, N.C.: ISA, 2001.

[71] H. Derhamy, J. Ronnholm, J. Delsing, J. Eliasson, and J. Van Deventer, "Protocol interoperability of OPC UA in service oriented architectures," in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, Emden: IEEE, Jul. 2017, pp. 44–50.

[72] M. Silveira Rocha, G. Serpa Sestito, A. Luis Dias, A. Celso Turcato, and D. Brandao, "Performance comparison between OPC UA and MQTT for data exchange," in *2018 Workshop on Metrology for Industry 4.0 and IoT*, Brescia: IEEE, Apr. 2018, pp. 175–179.

[73] A. Abdelsattar, E. J. Park, and A. Marzouk, "An OPC UA client/gateway-based digital twin architecture of a SCADA system with embedded system connections," in *2022 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Sapporo, Japan: IEEE, Jul. 11, 2022, pp. 798–803.

[74] OPC Foundation, *OPC 10000-1: UA Part 1: Overview and Concepts - 3.2.1 AddressSpace , v1.05.02*, `https://reference.opcfoundation.org/Core/Part1/v105/docs/3.2.1`, Oct. 2023.

[75] ——, *The Industrial Interoperability Standard*, `https://opcfoundation.org/about/what-is-opc/`, Nov. 2022.

[76] H. Haskamp, F. Orth, J. Wermann, and A. W. Colombo, "Implementing an OPC UA interface for legacy PLC-based automation systems using the azure cloud: An ICPS-architecture with a retrofitted RFID system," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, St. Petersburg: IEEE, May 2018, pp. 115–121.

[77] H. Raddatz, E. Mahmoud, F. Holzke, P. Danielis, D. Timmermann, and F. Golatowski, "Evaluation and extension of OPC UA publish/subscribe MQTT binding," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, Tampere, Finland: IEEE, Jun. 10, 2020, pp. 543–548.

[78] IEC TR 62541-1:2020, *Opc unified architecture - part 1: Overview and concepts*, Nov. 2020.

[79] T. Mizuya, M. Okuda, and T. Nagao, "A case study of data acquisition from field devices using OPC UA and MQTT," in *2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, Kanazawa: IEEE, Sep. 2017, pp. 611–614.

[80] OPC Foundation, *OPC 10000-1: UA Part 1: Overview and Concepts - 6.1 Client Server Overview, v1.05.02*, `https://reference.opcfoundation.org/Core/Part1/v104/docs/6.1`, Oct. 2023.

[81] ——, *OPC 10000-1: UA Part 1: Overview and Concepts - 6.2 OPC UA Clients, v1.05.02*, `https://reference.opcfoundation.org/Core/Part1/v105/docs/6.2`, Oct. 2023.

[82] ——, *Unified Architecture. Part 14: PubSub, v1.05.01*, `https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-14-pubsub`, Mar. 2022.

[83] ——, *OPC 10000-14: UA Part 14: PubSub - 5 PubSub Concepts, v1.05.02*, `https://reference.opcfoundation.org/Core/Part14/v105/docs/5`, Oct. 2023.

[84] ECLIPSE foundation, *4diac IDE - IEC 61499 Compliant Development Environment*, `https://www.eclipse.org/4diac/en_ide.php`, Jan. 2023.

[85] NetworkX developers, *NetworkX, Release 3.0*, `https://networkx.org/`, Jan. 2023.

[86] Inductive Automation, *Ignition SCADA by Inductive Automation*, `https://inductiveautomation.com/ignition`, Jan. 2023.

[87] Unified Automation GmbH, *UaExpert—A Full-Featured OPC UA Client*, `https://www.unified-automation.com/products/development-tools/uaexpert.html`, Dec. 2022.

[88] open62541, *Open62541*, `http://open62541.org`, Jan. 2023.

[89] Beckhoff Automation, *TF6100 TwinCAT 3 OPC UA*, `https://www.beckhoff.com/en-en/products/automation/twincat/tfxxxx-twincat-3-functions/tf6xxx-tc3-connectivity/tf6100.html`, Jan. 2023.

[90] Inductive Automation, *Ignition opc ua*, `https://inductiveautomation.com/ignition/modules/ignition-opc-ua`, Jan. 2023.

[91] Opcua-asyncio, *Opcua-asyncio*, `https://github.com/FreeOpcUa/opcua-asyncio`, Jan. 2023.

[92] Inductive Automation, *Ignition 8.1 manual. Scripting.* `https://docs.inductiveautomation.com/display/DOC81/Scripting`, Jan. 2023.