A Software Architecture to Control Service-Oriented Manufacturing Systems

Sebastiano Gaiardelli, Stefano Spellini, Marco Panato, Michele Lora, Franco Fummi
Dept. of Computer Science – University of Verona
name.surname@univr.it

***Abstract*—This paper presents a software architecture extending the classical automation pyramid to control and reconfigure flexible, service-oriented manufacturing systems. At the Planning level, the architecture requires a Manufacturing Execution System (MES) consistent with the International Society of Automation (ISA) standard. Then, the Supervisory level is automated by introducing a novel component, called Automation Manager. The new component interacts upward with the MES, and downward with a set of servers providing access to the manufacturing machines. The communication with machines relies on the OPC Unified Architecture (OPC UA) standard protocol, which allows exposing production tasks as "services".**

The proposed software architecture has been prototyped to control a real production line, originally controlled by a commercial MES, unable to fully exploit the flexibility provided by the case study manufacturing system. Meanwhile, the proposed architecture is fully exploiting the production line's flexibility.

***Index Terms*—Software architecture, Agile manufacturing, Manufacturing automation**

I. Introduction

Traditional production paradigms are no longer consistent with modern market requirements, and manufacturing technologies must evolve to cope with the increasing unpredictability of modern society conditions. "Industry 4.0" [1] is meant to assist this transformation, proposing a set of *production systems development guidelines* to a wide range of engineering disciplines. Among the promises of the Industry 4.0 trend, the concept of *reconfigurability* stands out as a key factor to quickly adapt the production to sudden market changes [2].

The reconfiguration of a production line is a multi-layered problem, bridging business aspects to automation control. Traditional monolithic information systems, currently used by most companies (*i.e.*, MESs) do not provide the necessary flexibility and agility to support efficient system reconfiguration. Service Oriented Manufacturing (SOM) is an emerging context in industrial automation [3], where systems distribute the responsibility of carrying out functionalities across the available manufacturing components. Nonetheless, the adoption of SOM-based technologies in a production environment is far from being an easy task, especially considering the reluctance of stakeholders to embrace new (and potentially production-breaking) technologies [4]. Specifically to the business level, getting rid of a monolithic MES in favour of a distributed architecture may not be a viable option.

This paper proposes a variation to the classic automation pyramid. It modifies the traditional software architecture by automating the supervisory layer, which becomes *Automated Supervisory*. The modified layer, as depicted in Figure 1, encloses the *Automation Manager*, which is composed of multiple software modules carrying out a specific functionality. In particular, it provides *Seamless integration* with existing software architecture, *Automated reconfiguration* of the production system, *Easy integration* of new technologies, *Autonomous execution* of production orders, *Resource management Advanced scheduling*, and support for *Data analysis*.

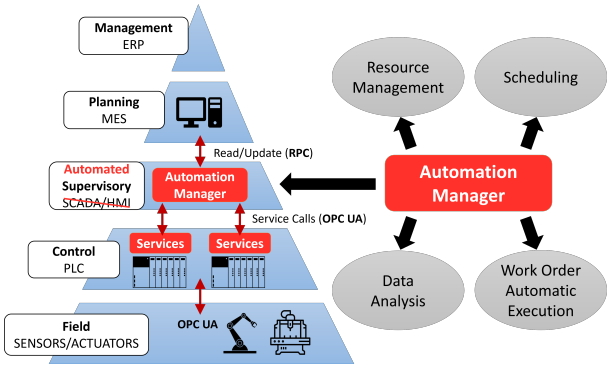

Figure 1: The classic *Automation Pyramid* (on the left, in blue), and the localization of the *Automation Manager* (in red) proposed in this paper, and modifying the software architecture.

The approach's flexibility is guaranteed by its compatibility with standard communication protocols (*i.e.*, OPC UA), to interact with services at Programmable Logic Controller (PLC) level. The adoption of the ISA-95 standard ensures terminology and data compatibility with commercial MESs, thus easing the adaptation of existing manufacturing systems.

To demonstrate the applicability and the efficiency of the proposed architecture, we integrated the Automation Manager into a real production line equipped with machines providing SOM features. Despite a little overhead, the advantages in configuration simplicity and software flexibility, provided by the proposed architecture, are relevant.

II. Preliminaries

The architectural model, followed by smart manufacturing and also referred to as the *Automation Pyramid*, is shown on the left of Figure 1. It consists of five layers with different structures, requirements, and temporal constraints:

- *Field level*: sensors, actuators, and applications that physically act on the production floor.
- *Control level*: PLCs gathering information from the sensors to control the actuators.
- *Supervisory level*: high-level supervision with Supervisory Control and Data Acquisition (SCADA) and Human-

This work is partially supported by the *European Commission through the project DeFacto (grant n. H2020-MSCA-IF-2019-894237)*, and by the *Veneto Regional project VIR2EM (grant. POR FESR Azione 1.1.4)*.

Machine Interaction (HMI) controlling multiple machines and collecting data from them.
- *Planning level*: a MES monitoring the manufacturing processes transforming raw materials into finished products.
- *Management level*: connects the production infrastructure with the Enterprise Resource Planning (ERP) system.

To integrate new technologies within manufacturing systems it is necessary to define a standard terminology and a unique data representation used throughout all the automation levels. ISA-95, also known as the IEC 62264 standard [5], aims at guiding the development and integration of manufacturing software. It defines functionalities, responsibilities, standard terminology, and data exchange between the enterprise and process control levels. This allows simplifying the integration of software within a manufacturing industry.

A. Communication Protocols

The *OPC UA protocol* plays a very dominant role in industrial applications: it became a de facto standard for Machine to Machine (M2M) communication in industrial automation. It is a platform-independent service oriented protocol standardized in IEC 62541 [6]. The communication is based on a client/server structure, where the server exposes the information model. Due to its versatility, OPC UA allows to model data transport compliant with the ISA-95 standard [7].

Distributed applications realizing manufacturing services need scalable, fault-tolerant and low-latency communication channels to interact with each other. To implement such requirements, *Apache Kafka* has been proposed. It is a distributed publish-subscribe messaging system developed to process real-time data with low latency [8]. Messages are stored as records including any kind of information, and they are stored until a specified retention period has passed. Kafka records are organized into topics, where producer applications publish new messages and consumers read the subscribed data.

While Kafka is best suited for low-latency and high throughput applications, *RabbitMQ* performs best when security and request-response messages are primary concerns [9]. It is a message broker supporting different messaging protocol, where producers publish messages into a queue. It also supports delivery acknowledgment and the possibility to assign permission such as rights to read and write to different users.

III. Services and Data Collection Architecture

We propose a modular SOM-enabled architecture, located in a Kubernetes cluster connecting the automation level to classical MES solutions. An overview of the proposed architecture's structure is provided in Figure 2. It consists of different applications, communicating with each other through Kafka and RabbitMQ. The *Automation Manager* manages the communication with the MES while extending its functionalities to add reconfiguration of the production line, autonomous execution of production orders and advanced scheduling. This section describes the architecture components; Section IV will deep dive into the Automation Manager.

A. OPC UA Servers

Each machine in the system is equipped with an *OPC UA server* module. The piece of equipment and its server module are strongly intertwined. The data model (in particular the machine status and sensors data) exposed by the server depends on the functions developed to externally control the machine. To create the *OPC UA server* module, the machine's functions are wrapped and exposed as *OPC UA methods*. Then, the *OPC UA data model* is enriched with state variables that a client can read to know the status of the running operations.

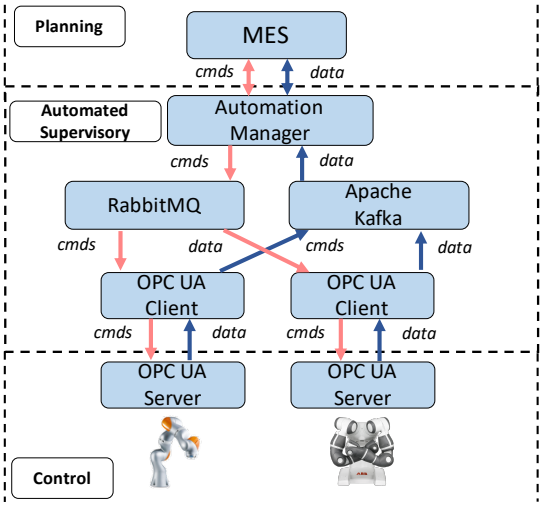

Figure 2: Automation of the *Supervisory level* by introducing the proposed service-oriented architecture. Arrows show the commands and data flow through the automation pyramid and our novel architecture.

B. Data Collection Infrastructure

An Industrial Internet of Things Data Collection Infrastructure monitors the connected equipment and stores the gathered data. It features a multi-node *Apache Kafka* instance handling data streams. Meanwhile, a multi-node *RabbitMQ* instance is in charge of handling remote procedure calls through queues.

The Data Collection Infrastructure communicates with the equipment through multiple *OPC UA Client* nodes: an instance (and configuration) is active for each *OPC UA server*. This node creates a persistent connection with the machine and creates an *OPC UA subscription*, specifying the *OPC UA variables* to monitor. Each time a variable changes, the client is notified with the new value, which is written to the configured Kafka topic. The *OPC UA client* nodes are also connected to the RabbitMQ instance and listen for Remote Procedure Call (RPC) requests from the configured queues. Allowed requests for this client are: the *read* of variable, the *write* on a variable, or the *invoke* executes an *OPC UA method*.

Figure 2 shows the data and command flows into the proposed architecture. Each machine has its own Kafka topic and RabbitMQ RPC queue, managed by the corresponding OPC UA client instance.

IV. Automation Manager

To handle the communication with the MES and the machines transparently, the *Automation Manager* is organized in three different layers and many sub-components, depicted in Figure 3. The top layer contains the software *Driver* interfacing the Automation Manager with the upper layers of the automation pyramid; the bottom layer is the software *Driver* connecting the manager with the lower layers of the

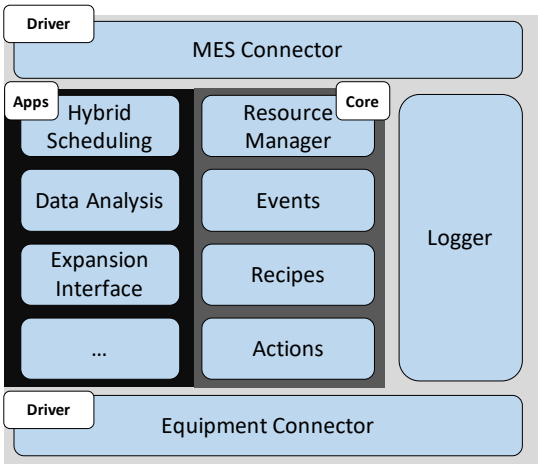

Figure 3: The internal structure of the *Automation Manager*.

pyramid; the middle layer contains the manager *Core*, a set of *Applications*, and a *Logger*. The Automation Manager is compliant with ISA-95 standard. Thus, it is compatible with any existing software infrastructure based on the same standard. It takes in input information coming from both the MES, which contains the production line structure and recipes, and from a set of configuration files describing the machine capabilities and recipe implementations. These characteristics are meant to ease as much as possible its integration within already existing manufacturing plants.

A. Drivers

The *Driver* levels contain the components enabling the communication with other pieces of software. The *Equipment Connector* exposes for the other levels basic functionalities of OPC UA, such as variable read/write, methods call, and subscriptions for data changes. It also communicates with the OPC UA clients connected to the machines within the cluster. The *MES Connector* is implemented as an RPC client calling functions defined by an RPC server connected directly with the MES. As an advantage of using RPC interfaces, the integration with any other MES only requires to implement the corresponding RPC server. This driver allows navigating through the MES configuration and notifying the actions executed by the architecture, such as execution of operations and reconfiguration. The last driver is the *Logger*. It publishes log messages on two different Kafka topics: one consists of messages useful for debugging purposes; the other includes messages logging actions executed by the architecture, such as the execution of recipes or machine services. This allows notifying the entire architecture of the status of each level.

B. Core

The second level contains the *Core* components, defining and implementing industrial processes. The MES represents production recipes as a sequence of dependant tasks, each associated specifically to a *class* of working cells (*i.e.*, a work center). On one hand, this allows to model production processes at a higher level of abstraction, hiding unnecessary implementation details. On the other hand, this is not enough to execute tasks without human intervention. Therefore, the recipe representation in the Core of the Automation Manager is extended with a lower-level model that describes the implementation of tasks on the working cells. This representation consists of an ordered sequence of actions with input and output parameters, formalized as a directed cyclic graph: the actions are nodes of the graph, connected by directed edges to represent dependencies. An action can be a service exposed by the Equipment Connector or a logical construct (*i.e.*, creation of variables, the sum of variables, if, cycle, *etc.*) proposed by the Core. This extension allows executing tasks with a simple visit of the graph nodes. Then, the actual execution of tasks is managed by the *Resource Manager*. It retrieves the manufacturing structure from the MES and, for each working cell, it connects to the correct machine's client. This ensures that when an operation is executed on a working cell, it has access solely to those clients. It also guarantees that a maximum of one operation is executed on a working cell at the same time.

C. Applications

The scheduling of production processes on different machines in a dynamic environment is still an open problem. In the literature, this problem is known as Dynamic Flexible Jobshop Scheduling (DFJSS). Although there are many solutions to the static counterparts of this problem, known as Jobshop scheduling (JSS) and Flexible JSS (FJSS), when we introduce the dynamic component that characterizes real systems the solutions for these problems are not applicable. The reason is that every time an unexpected event occurs (*i.e.*, the arrival of new orders, machine breakdowns and delays), the schedule is no longer optimal or even not feasible anymore. Therefore, a promising direction is to introduce static-reactive scheduling, characterized by a first phase that produces a static schedule of the jobs, dynamically updated on the arrival of events [10]. In the first implementation of the proposed architecture, we opted for this hybrid approach to implement scheduling. It consists of a static phase exploiting constraint programming to produce an optimal solution while minimizing energy consumption and delays. Then, a dynamic component continuously recalculates the scheduling to react whenever an unexpected event occurs, such as new job arrivals and machine breakdowns.

Furthermore, to achieve more precise scheduling, we developed a data analysis application receiving timing data about executed production processes. The gathered data is used to update the completion time estimation of production processes. To support the integration of applications based on different technologies, an expansion interface exposes the functions of the SOM-enabled architecture's core.

V. Experimental Results

We evaluate the proposed architecture in the Industrial Computer Engineering (ICE) Laboratory: a research facility of the University of Verona[1], equipped with a complete manufacturing line [11]. The line is governed by a state-of-the-art system implementing the automation pyramid, centered around a commercial, monolithic MES offering some advanced features. The machines expose services through OPC UA servers, and the MES is orchestrating the execution of the different processes. Production orders are manually executed by operators,

[1]The ICE Laboratory: https://www.icelab.di.univr.it/

Table I: Comparison of functionalities available when using the traditional software stack against the proposed SOM-enabled architecture.

Functionalities	Traditional Pyramid	Automated SOM
Data collection	✓	✓
Product and Process monitoring	✓	✓
Inventory tracking	✓	✓
Advanced production planning	✓	✓
Resource management	✓	✓
Automatic reconfiguration		✓
Autonomous process execution		✓
Integration of new software modules		✓
Run-time adaptive scheduling		✓
Reduced configuration time		✓
Reduced deployment time		✓

Table II: Comparison between the communication delay derived from a direct connection with OPC UA and with the proposed architecture.

Transport Type	Read (s)	Write (s)	Methods (s)	Subscription Update (s)
OPC-UA	0.008	0.009	0.010	0.150
SOM	0.013	0.013	0.014	0.255
Overhead	62.50%	44.45%	40.00%	70.00%

and features such as reconfiguration or advanced scheduling are not available. We implemented and deployed the proposed architecture on the ICE laboratory production line. We first defined and implemented the Kubernetes cloud architecture. Then, we configured and deployed the Automation Manager. In our tests we use three different complete production recipes. First, we evaluate the proposed architecture qualitatively, by analyzing the newly available features. Then, we evaluate it quantitatively by measuring the introduced overhead.

A. Qualitative Analysis

Table I summarizes the functionalities implemented by the traditional architecture compared to those provided by the presented solution. Among the new features introduced, the Automation Manager reduces the effort necessary to configure and reconfigure the entire software architecture, from the low-level task implementation to the high-level recipe representation and machine structure. This allows creating a flexible implementation of productive processes, based on actions (*i.e.*, services) exposed by the machines and recipes specified within our novel architecture as a sequence of these actions. It also introduces the possibility to adapt the realization of production processes at run-time, to minimize the total execution time or to pursue a specific production objective. Furthermore, the hybrid scheduler continuously adapts the production plan as a response to unforeseen events and manages the execution of processes on the entire plant. Therefore, it reconfigures the system by automatically changing the sequence of manufacturing operations during a certain time frame.

B. Overhead Analysis

Additional features come at a price in terms of computational overhead: Table II reports the overhead necessary to call OPC UA functions comparing a direct connection with the machines and through the proposed architecture. We compared the delay of different services, such as read/write of variables, method calls, and subscription to variables. For each operation, the last line reports the additional overhead (in percentage) required when using the proposed architecture.

Table III: Execution time when using the state-of-the-art versus the proposed architecture to govern three different production recipes.

Recipe	Tasks	Service Calls	OPC UA Time (s)	SOM Time (s)	SOM Overhead
1	4	54	70.34	70.85	0.72%
2	5	44	66.73	67.04	0.46%
3	11	132	158.83	159.63	0.50%

The additional overhead introduces a communication delay ranging between 40% and 70%. However, this communication delay is in the context of complex physical processes. For this reason, we also evaluated the behavior of the proposed architecture when coordinating different manufacturing processes. Table III reports the total execution time for three production recipes of different sizes, comparing the state-of-the-art architecture with our proposed solution. The total execution times do not consider the transportation time required to move materials on the conveyor belts, as it is influenced by many factors independent on the control architecture. The fourth and fifth columns of the table report the execution times obtained with the two different configurations. The last column reports the overhead introduced by the proposed architecture. The delay introduced is minimal and consequently negligible from the total execution time. Finally, comparing Tables II and III, it is worth noticing that while the additional overhead is significant when considering single operations, it becomes negligible in the context of a complete manufacturing process.

VI. Conclusions

We presented a variation of the traditional automation pyramid software stack to control SOM systems. The results of the application of the proposed architecture to a real production plant showed that the overhead introduced is negligible. Nonetheless, the added functionality over state-of-the-art architecture proved to increase production flexibility.

References

[1] R. Drath and A. Horch, "Industrie 4.0: Hit or hype? [industry forum]," *IEEE Industrial Electronics Magazine*, vol. 8, no. 2, pp. 56–58, 2014.

[2] Y. Koren *et al.*, "Reconfigurable Manufacturing Systems," *CIRP Annals*, vol. 48, no. 2, pp. 527–540, 1999.

[3] T. Lojka, M. Bundzel, and I. Zolotová, "Service-oriented architecture and cloud manufacturing," *Acta polytechnica hungarica*, vol. 13, no. 6, pp. 25–44, 2016.

[4] K. Tiwari and M. S. Khan, "Sustainability accounting and reporting in the industry 4.0," *Journal of Cleaner Production*, vol. 258, 2020.

[5] International Society of Automation, "ISA-95 Standard," 2000. [Online]. Available: https://www.isa.org/

[6] "OPC Unified Architecture specification – Part 1: Overview and concepts release 1.04 OPC Foundation," 2017.

[7] M. V. García *et al.*, "From ISA 88/95 meta-models to an OPC UA-based development tool for CPPS under IEC 61499," in *Proc. of IEEE WFCS*, 2018, pp. 1–9.

[8] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011.

[9] P. Dobbelaere and K. S. Esmaili, "Kafka versus RabbitMQ: A Comparative Study of Two Industry Reference Publish/Subscribe Implementations: Industry Paper," in *Proc. of ACM DEBS*, 2017, p. 227–238.

[10] O. Cardin *et al.*, "Coupling predictive scheduling and reactive control in manufacturing hybrid control architectures: state of the art and future challenges," *Journal of Intelligent Manufacturing*, vol. 28, no. 7, 2017.

[11] S. Spellini *et al.*, "Virtual Prototyping a Production Line using Assume-Guarantee Contracts," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, 2020.