Non-invasive software architecture for data pipelines with legacy support in smart manufacturing

Conference Paper · March 2025

DOI: 10.1109/ICSA65012.2025.00034

3 authors:

Alberto Ceselli
University of Milan
105 PUBLICATIONS **1,744** CITATIONS
SEE PROFILE

Giuseppe de Martino
University of Milan
11 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Patrizia Scandurra
University of Bergamo
143 PUBLICATIONS **2,178** CITATIONS
SEE PROFILE

Non-invasive software architecture for data pipelines with legacy support in smart manufacturing

Alberto Ceselli
Università degli Studi di Milano
Milano, Italy
alberto.ceselli@unimi.it

Giuseppe De Martino
Università degli Studi di Milano, Par-Tec S.p.A.
Milano, Italy
giuseppe.demartino@{unimi.it,par-tec.it}

Patrizia Scandurra
Università degli Studi di Bergamo
Bergamo, Italy
patrizia.scandurra@unibg.it

***Abstract*—Smart manufacturing relies on the digitization of all the industrial processes, from production to business operations. It uses Industrial Internet of Things (IIoT) principles to equip devices with smart sensors and actuators, integrating machines and software through data collection, advanced computational methods, and remote control. Our research is motivated by a real digital transition application in the luxury fashion in Italy. The customers wish to update legacy systems, to comply with new Industry 4.0 standards. Due to industrial property requirements, as well as brand secrets, they require the whole architecture to run on-premises. A further requirement is that the system installation must be non-invasive, potentially running on systems with frugal setups in terms of hardware and software.**

Adhering to such requirements and principles, this paper proposes an architecture for data pipeline in smart manufacturing that runs on-premises, offering support to legacy machines. It is capable of identifying unknown hardware, in terms of semantics of its sensors. The core component of such a concrete architecture is an innovative Extract-Transform-Load (ETL) connector, called sEmantic eXtended ETL (exETL), that manages numerous heterogeneous data sources, and recognizes and configures automatically new machinery sensors. It employs a dedicated Machine Learning (ML) pipeline. The flexibility of the proposed architecture is compared to alternative solutions that exploit existing technologies. Its computational effectiveness is assessed by building an emulated environment, and running extensive experiments on real data. Our results show that our data pipeline is lightweight, more flexible than competitors, and capable of integrating legacy or new machinery seamlessly.

***Index Terms*—Software architecture for Smart Manufacturing, Industry 4.0, Data Pipeline, semantic ETL, unknown sensors**

I. Introduction

Industry 4.0 refers to the ongoing transition of traditional manufacturing and industrial practices into smart manufacturing through the integration of digital transformation technologies such as Industrial Internet of Things (IIoT) for remote sensing and actuation, augmented reality solutions, AI/ML and data analytics. In more advanced scenarios, interconnected cyber-physical systems (such as intelligent robots and digital twins) can self-diagnose possible failures in the manufacturing system, warn, and self-heal themselves. This transformation aims to create more efficient, flexible, responsive, and reliable manufacturing processes, from the factory floor to all aspects of business (product design, product production, distribution, supply chain, logistics, sales, etc.).

The transition to smart manufacturing starts with bringing analog data into a digital database. However, collecting the data and engineering a data pipeline purposefully is still a complex task [1]. Moreover, there is also an increasing attention toward non-invasive installations that minimize changes on the on-site manufacturing system. In textile-fashion sector and textile machinery sector in Italy, for example, it is common to find a variety of both legacy and modern machines supplied by different vendors (not necessarily made-in Italy), each adopting distinct data encoding protocols. Manual configuration of a new machine is often required, which can occur at any level of the infrastructure, from the Programmable Logic Controller (PLC) to the Data Warehouse (DW) server. In addition to the legacy support and disparate data sources and formats, high-tech companies (as those in high-quality fashion) are often concerned with protection of industrial property and brand secrets. Such a need leads to a strong additional requirement: to run all data processing on-premises.

To accommodate all these requirements at once, this paper proposes a non-invasive architectural solution for a data pipeline in smart manufacturing, running on-premises with a frugal hardware and software setup, and able to automatically adapt to new machinery, in terms of configuration of unknown sensors. The core component of our solution is a novel Extract-Transform-Load (ETL) process. An ETL process in a data pipeline typically transforms and organizes raw data according to some business logic, storing them into a single repository (e.g., a DW), optimized for analytics. We propose a *sEmantic eXtended ETL* (exETL) capable of managing multiple sources, generating information in heterogeneous formats, and ensuring the quality of the data [2] populating the DW. In addition to a conventional ETL, exETL can manage potentially unknown data sources, such as new machinery sensors, by identifying their specific categories through a dedicated Machine Learning (ML) pipeline [3].

The proposed architectural solution is the result of a joint

The work was partially supported by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, and by the Research Support Plan of the *Università degli studi di Milano*.

2835-7043/25/$31.00 ©2025 IEEE
DOI 10.1109/ICSA65012.2025.00034

collaboration with a company[1] that has direct experience in the field and has Italian high fashion textile companies as clients.

A conceptual preview of such a solution is shown in Fig. 1 by considering the three tiers of a conventional IIoT-Edge-Cloud architecture for smart manufacturing. The *Physical level* is made of legacy industrial machines (IIoT) equipped with sensors, actuators, PLCs, and additional hardware components that receive commands from PLCs. The *Edge level* hosts an intermediate computation layer of micro-computers that perform data collection from sensors of various connected machines and transformation. Sensors data is transmitted from the machines to a message broker, and exETL runs at this level too, by reading, transforming, and loading data into DBMSs. However, it can happen that a company acquires machinery from different suppliers, which means that, despite generating the same data stream, this data is saved in different PLC register addresses compared to those of existing machinery. Consequently, the middleware responsible for data extraction may not be able to recognize the register and thus sends all the data found in the register to the message broker. In this scenario, thanks to its sensor recognizer functionality, exETL is able to identify the topic of sensor data, and configure the sensor by associating a proper transformation rule. Lastly, the *Cloud on-premises level* comprises a server/or a private Cloud located within the company, where various DBMSs forming the data warehouse and all software applications for managing business processes are installed – the Manufacturing Execution System (MES), the Enterprise Resource Planning (ERP), and the Decision Support System (DSS).

The main contributions of this work are therefore:

- a concrete architecture specifically targeting the high-end textile industry, and aimed at transforming textile companies from traditional industries to Industry 4.0 following *principles of non-invasiveness* for data collection and transformation;
- a novel ETL process, exETL, running in edge and able to accommodate transformation of unknown sensor data via a dedicated ML classification;
- an experimental evaluation on real data showing the effectiveness, efficiency, and flexibility of exETL.

This paper is organized as follows. Section II provides some preliminary concepts about the ETL process. Section III reports on related works. Section IV describes the proposed architectural solution. Section V details the exETL connector. Section VI presents the results of an empirical evaluation of exETL. Section VII discusses threats to validity and Section VIII collects some brief conclusions.

II. Background

This section provides preliminary concepts about the ETL data flow process and types of ETL in smart manufacturing.

[1]The subject ICT company is Par-Tec S.p.A.. In the area of Industry 4.0, at the time of writing, such ICT company has signed agreements with more than 50 companies in the textile sector, converting more than 580 legacy machines into Industry 4.0-compatible ones. Each customer company is equipped with at least one edge node running our ETL solution, for a total of more than 72 edge nodes spread across all customers.

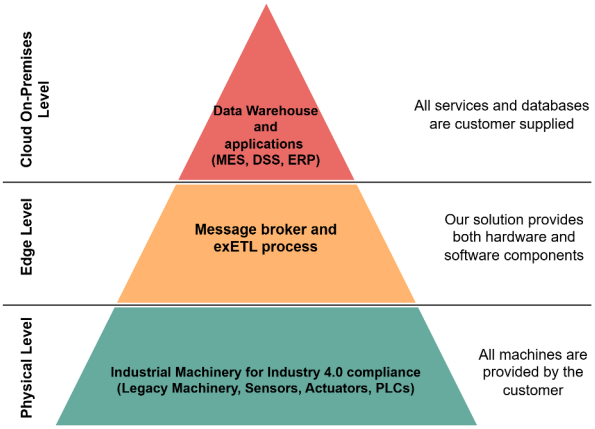

Fig. 1. High level representation of the levels of automation in a factory Industry 4.0 as covered by our exETL process

ETL is one of the most popular approaches for data integration. The ETL process has experienced a significant increase and is employed in multiple business contexts, adapting to various technologies. It is used to extract data from various sources and populate different types of data warehouses, both SQL-based and NoSQL-based [4]. Another application of ETL involves extracting data that is continuously generated, and its utilization is fundamental for real time data stream analysis. In [5], for example, a framework for an ETL process is proposed, to populate a document-oriented NoSQL data warehouse, providing real-time analytics.

According to a recent study [6], three categories of ETL exist across three operational contexts: *Business Intelligence* (BI) for data analysis to support strategic decision-making and improve performance, *Big Data* context for managing large volumes of unstructured and semi-structured data, and *Semantic* context for identifying data in a way that is understandable and interconnected, enhancing interoperability between different systems.

Our exETL covers all three contexts.

BI context: It consists of the traditional application of ETL, focusing primarily on data collection and cleansing, with the aim of storing them in a multidimensional database to feed decision support applications.

All our Industry 4.0 clients utilize sector-specific applications such as MES and DSS, which necessitate a suitable data transformation tool for the proper utilization of sensor data from textile machinery.

Big Data context: Due to the complexity of their products, manufacturing companies, especially in the textile sector, use various types of industrial machinery equipped with multiple sensors that generate large amounts of data in a very short time. In this context, the ETL is primarily aimed at harnessing data characterized by 5 V's of Big Data:

Volume Sewing machines gather various data points like

needle speed and thread tension rapidly, while industrial ovens focus on slower-paced data such as temperature monitoring.

Velocity Sewing machines transmit data rapidly, especially during production peaks, while ovens have a slower, more consistent transmission rate due to less frequent changes in temperature and cooking parameters.

Variety The textile production chain relies on a wide range of industrial machinery, each generating specific data types related to its function.

Veracity In luxury fashion textiles, to trust the information used to reach decisions the considered clients invest in industrial machinery from reputable IIoT suppliers, known for installing precise and reliable sensors.

Value It is the added-value that the collected data can bring to the intended process, activity or predictive analysis. Analyzing data from textile machinery may be used, for example, to optimize production processes, reduce waste, improve product quality, and prevent unplanned production interruptions.

Semantic context: This type of ETL extends traditional ETL by taking into consideration semantic data. It is based on ontologies (*Semantic* ETL or ETL based on ontologies) to tackle integration issues related to data heterogeneity, including structured, semi-structured, and unstructured data, as well as semantic heterogeneity and data inaccessibility for users. Using semantic and knowledge management technologies, this ETL harmonizes heterogeneous data from multiple sources, thus providing a common, unified data repository.

In the textile sectors, companies are equipped with heterogeneous machinery, often coming from different suppliers even if of the same type. This machinery heterogeneity results in a highly varied flow of sensor data. In some cases, this data is structured and comes from more advanced machinery, while in other cases, it is raw data in a binary and unstructured format. So this sector would benefit a lot from a Semantic ETL.

III. Related work

A. Software architectures in smart manufacturing

Due to the highly heterogeneous data and processes driven by specific problems or requirements, no single defined software architecture model exists for a manufacturing system that perfectly aligns with the diverse needs and industrial experiences of each manufacturing company. However, over the last decades, some reference architectures for Industry 4.0 have been proposed and identified for the manufacturing sector, as emerged from the review studies in [7], [8]. According to the literature, the main models, selected for their level of maturity and acceptance, range from conceptual ones, such as RAMI 4.0 and IVRA, to more detailed architectures (e.g., IIRA, SITAM, IBM Industry 4.0, and LASFA, to name a few). Not all of them cover all five levels of the traditional and widely understood concept of *manufacturing automation pyramid* [9], [10] that shows how devices, actuators and sensors on the factory floor are distinctly separate from the process control, supervisory networks and enterprise systems. These levels (from the base of the pyramid) are: 1. *Field Level*, Industrial machinery with sensors and actuators; 2. *Control Level*, reads data from sensors and acts through PLCs; 3. *System/Process* Level, manages field devices and processes data in real-time; 4. *Line Level*, software systems like MES monitor the entire production process; 5. *Company Level*, applications like ERPs manage business processes.

Some architecture models only offer software, covering just the Line and Company levels, while others operate at the Field and Control levels [7]. With the proposed concrete architecture for the textile manufacturing sector, we mainly operate at the Control and System/Process levels, through customizations and integrations of the manufacturing system already in use at the Line level[2]. This is also our main requirement for the commencement of the work at a manufacturing company: modernization of the legacy support is to be done with low invasiveness and minimal disruption. To the best of our knowledge, this aspect is overlooked in these existing reference architecture models.

B. Enabling technologies for data pipeline

Numerous commercial platforms are available for managing data streams and building ETL processes. They provide high performance and intuitive dashboards for monitoring data movement. Among them, *NiFi*[3], *Kafka*[4], *Flink*[5] and *Spark*[6], all provided by Apache Software Foundation (ASF), stand out.

NiFi is a data integration tool for automating the flow of data between systems, but it is less effective for complex ETL processes. It is not designed for executing ML models directly, but can orchestrate data flows to externally hosted models.

Kafka is a distributed streaming platform optimized for real-time data streaming and event-driven architectures. Like Nifi, there is no native ML support and elaborated transformations and ML analysis require integration with external platforms through parallel processing libraries like *Kafka Streams*.

Flink is a distributed processing engine for stateful computations on both bounded and unbounded data streams. It offers high scalability, low latency, and high throughput, with strong guarantees like exactly-once state consistency and event-time processing. This makes it well-suited for data pipelines and ETL processes. However, the overhead of managing state and ensuring fault tolerance can introduce latency, which might not be ideal for real-time analytics at the edge. Moreover, Flink does not have built-in storage systems and relies on external systems (like HDFS or Kafka) for data storage and ingestion. This can also complicate deployments at the edge.

Spark is a multi-language engine for big data processing and analytics on both single-node machines and clusters. It

[2]With the adoption of Internet technologies, the automation pyramid, which originally had up to six distinct levels [10], tumbled down to three levels (as shown in Figure 1).

[3]https://nifi.apache.org/

[4]https://kafka.apache.org/

[5]https://flink.apache.org/

[6]https://spark.apache.org/

provides native support for in-memory distributed processing and fault tolerance, allowing complex data pipelines to be built. However, as Flink, running Spark on edge nodes can be cost-prohibitive due to the need for powerful hardware resources and high operational costs. Moreover, it is optimized for batch processing and may introduce latency in real-time analytics, which is crucial for edge computing scenarios.

Data processing and analytics on edge nodes typically need simpler, more lightweight and flexible solutions. Differently form the ASF tools mentioned above, the proposed exETL is a custom solution developed in Python and offers high flexibility in terms of data transformation and hosting of ML models by leveraging a wide range of libraries (such as Pandas, NumPy, scikit-learn, and TensorFlow, to name a few). These libraries enable complex data series transformations and easy adaptation to specific project requirements. A more detailed comparison among all these tools is reported in Section VI.

A recent study [11] provides an overview of non-custom solutions (COTS - commercial off-the-shelf) and custom solutions (like our approach). The survey examines the use of data-driven technologies in various phases of smart manufacturing (data ingestion, communication, storage, data analysis, and data visualization). It states that custom solutions for data ingestion are used in 47.3% of cases to handle proprietary protocols and complex data, so demonstrating they are equally important and necessary as non-custom solutions.

IV. Proposed software architecture

Fig. 2 shows a typical deployment of our concrete architecture – topology of the processing nodes (hardware) and software components (software & middleware) that run on them – reflecting the Industry 4.0 levels depicted in Fig. 1. Unlike the conventional cloud-computing model, such a deployment provides a continuum of computing from the IIOTs to the edge to the (private) cloud/ on-premises data centers – the *Edge-Cloud Continuum* (ECC) [12], [13] – to execute data (pre-)elaboration and control functions on the edge. Moreover, according to the design principle of *low invasiveness*, the edge nodes are the only infrastructure elements to add to the existing manufacturing system, while the infrastructure and systems at physical and cloud levels are provided by the customer.

Essentially, the manufacturing system comprises four main computing subsystems (see Fig. 2): the `Industrial Machinery` (the physical level) that encompasses a wide range of equipment (sensors, actuators, PLCs, and legacy machines) designed to perform specific tasks within production processes; the `Smart Interface` that implements machine-to-machine communication closer to the data source, the `Edge Node` that gathers, analyzes, and act on data across factory floor; and the `Application Server` that hosts the on-premises MES.

The `Smart Interface` and the `Edge Node` consist of a single-board computer (SBC), typically a Raspberry Pi 5, running the Linux Ubuntu operating system. The `Smart Manufacturing Agent` (`smAgent`) (a software module written in Python) is deployed on the `Smart Interface`, and is responsible for communicating via the protocol `OPC UA ModBus` to its PLC to extract sensor data and actuate commands over them accordingly. Additionally, the `smAgent` sends the sensor data via up-link messages to the broker hosted on the `Edge Node` using the Pub/Sub protocol MQTT (Message Queue Telemetry Protocol); data from a machinery's sensor is published on the topic `/idMachinery/nameSensor`, while data from unknown sensors is published on the topic `/unknownSensor`. The `smAgent` is also subscribed to the topic `/idMachinery/actuator` to receive back from the broker down-link commands to be executed on the industrial machinery. On the edge node, our exETL component is also installed. It subscribes to the message broker on the edge to read sensor data, transforms them using semantic transformation rules, and stores them in a storage located on the `Application server`. The `Edge Node` also contains a catalog `RuleTransformationCatalog` of all the semantic transformation rules that the exETL can apply. Each topic, linked to a sensor, is associated with a transformation rule. This catalog is pre-configured by the administrator for a specific installation. It is implemented as an XML configuration file conforming to a Web Ontology Language (OWL) document (or simply ontology). More details are given in Sect. V-C. The `Application Server` is typically Linux-based (e.g., Ubuntu Server) and is provided by the customer. It hosts a DBMS for data storage (predominantly MySQL), a software dashboard used for monitoring work processes and the status of machinery, and the MES for managing and controlling production operations.

The need to add a set of transformation rules for sensor data for each industrial machine arises from the fact that companies tend to expand their machinery fleet. By using our architecture, each time a new machine is added, the transformation process no longer requires modifications: it is sufficient to associate the machine with the pre-configured set of rules. However, it can happen that a company acquires machinery from different suppliers, which means that, despite generating the same data stream, this data is saved in different PLC register addresses compared to those of existing machinery. Consequently, the middleware responsible for data extraction may not be able to recognize the register and thus sends all the data found in the register to the MQTT broker. In this scenario, thanks to exETL that works also as sensor recognizer, an automatic process is triggered, which identifies the specific sensor, associates the correct transformation rule, configures the new sensor, and connects it to the new machinery.

V. exETL modeling and engineering

In the following we detail the behavior of the exETL connector. We combine its formalization with a running example.

A. exETL context

The operational context of exETL consists of the following entities:

- a set M of production *machines*

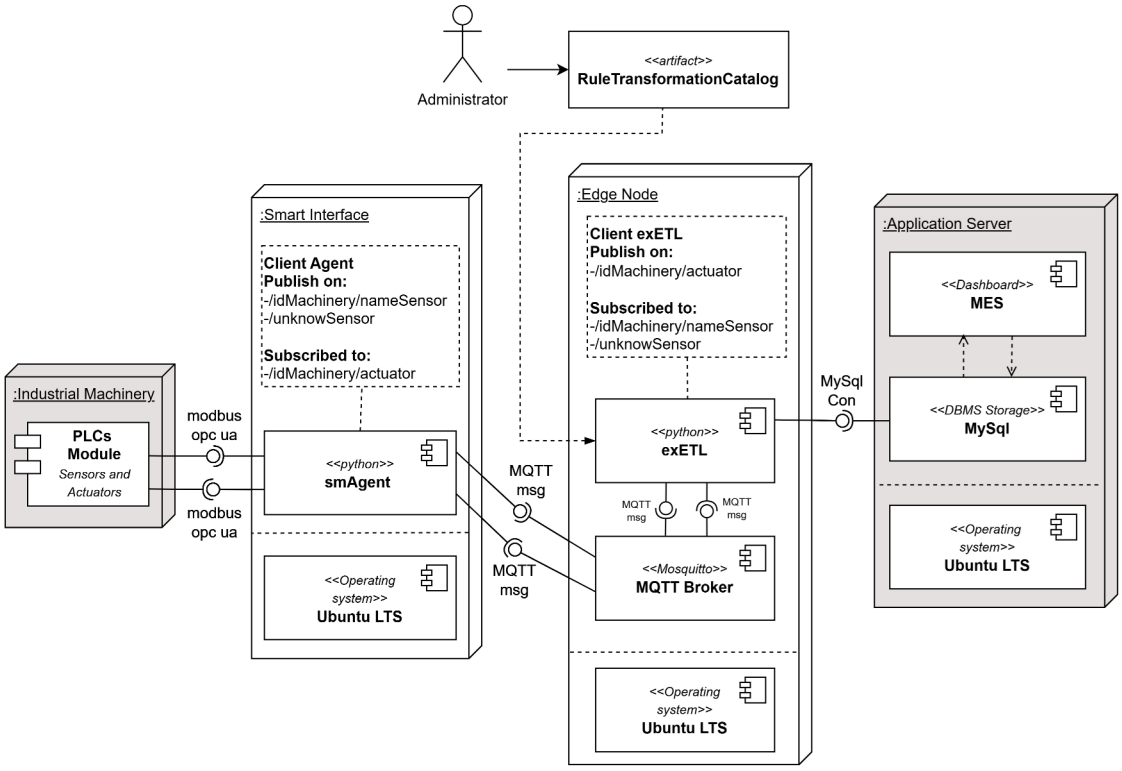

Fig. 2. Proposed architecture with exETL for transition to smart manufacturing

- a set S of data stream from *sensors*
- a set T of *topics* describing the possible semantics of data streams.
- a set R of *transformation rules* describing how data of each topic needs to be processed.

Each rule $r \in R$ corresponds to a pair $(t \in T, f())$: $f()$ is a transformation function to apply when data for t is produced.

For instance, a manufacturer may have two machines in production: one cutting and one sewing machine. Each machine has two sensors. The cutting machine is I4.0 ready. Its equipment is fully documented; it is therefore known that two sensors measure and stream temperature and pressure data, respectively. The sewing machine, instead, is a legacy one. IIoT devices have been installed on it as an upgrade. Therefore, it is not fully documented; it is also equipped with two sensors: one is known to be streaming the temperature of the machine, while the topic of data of the second one is initially unknown. From the machine type, we know it could be either pressure, rotor stop count, or an anomaly flag. We denote the mapping of sensor to topics as a function $\tau : S \to T$.

In this case, $M = \{\text{cutting, sewing}\}$, $S = \{\text{cutting-S1, cutting-S2, sewing-S1, sewing-S2}\}$, $T = \{\text{temperature, pressure, rotor, anomaly, T02-b}\}$. Furthermore, $\tau(\text{cutting-S1}) = \text{temperature}$, $\tau(\text{cutting-S2}) = \text{pressure}$, $\tau(\text{sewing-S1} = \text{temperature}$, $\tau(\text{sewing-S2}) = \text{T02-b}$.

A set of transformation rules can be the following: $R = \{(\text{temperature}, \text{float32}()), (\text{pressure}, \text{float64}()), (\text{rotor}, \text{int}())\}$, that is the bits of data are transformed with different encodings, based on the type of the sensor. Topic T02-b remains *unknown*, having no transformation rule.

The role of our exETL is to act as a traditional semantic ETL, applying transformations to sensor streams. However, instead of simply dropping data that has no transformation rule (as would happen to data of sensor sewing-S2), this data is logged and feeds an ML pipeline that attempts to detect if the unknown topic is matching a known one. In our running example, data from sensor sewing-S2, having no transformation rule, is kept in a log; with a certain frequency, the ML pipeline is invoked, possibly detecting the actual topic to be rotor.

When the ML pipeline detects the topic of an unknown sensor, it enlarges the set of rules with a new one, handling it. In our example, a new rule $\bar{r} = (\text{T02-b}, \text{int}())$ would be created and the set of rules would be updated as $R = R \cup \{\bar{r}\}$. Accordingly, the sensor starts having a suitable transformation rule; its data is not marked as unknown anymore: instead of logging, this data is processed by the ETL as those of the other sensors.

B. exETL workflow

The exETL workflow consists of three main layers: the `Semantic layer`, the `Buffering layer`, and the `ML`

`pipeline layer`. Figure 3 details the main steps of these stages using a UML activity diagram.

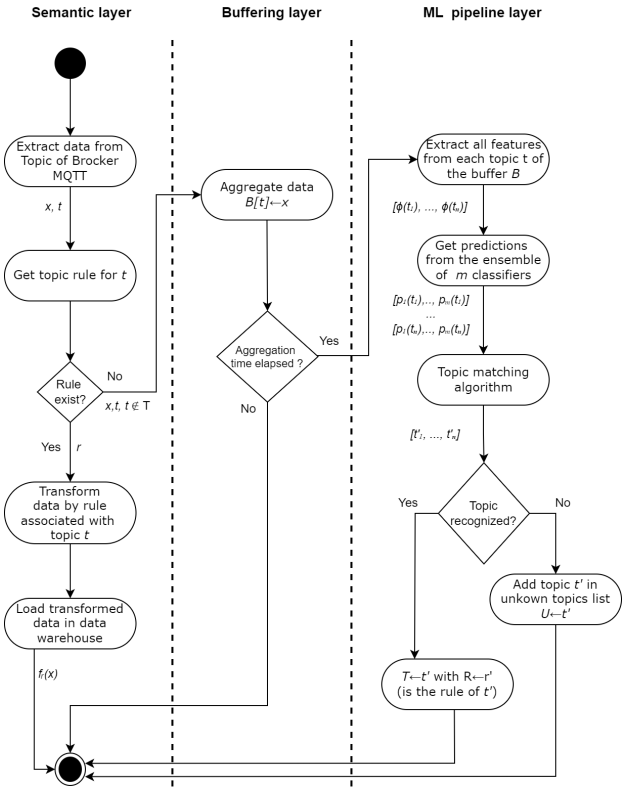

Fig. 3. Activity flow of the exETL

The process starts with the `Semantic layer` and behaves like a traditional semantic ETL [14] [6]. It is in charge of *extracting* the sensors data from the broker MQTT, performing *transformations* such as aggregation of values over time, and *loading* the transformed data into the company data warehouse. Transformations depend on topics. For instance, data about temperature and pressure may need to be normalized, while data from rotor stop count and anomaly may need to be aggregated into a frequency value. The specific transformation is therefore formalized by the set R of rules, as described previously. The exETL implements for each sensor $s \in S$ a *conditional function* $y = h_s(x, R)$, transforming input data x into output data y according to the applicable transformation rules R:

$$h_s(x,R) = \begin{cases} f_r(x) & \text{if } \exists r = (t_r, f_r()) \in R : t_r = \tau(s) \\ p_s(x) & \text{otherwise} \end{cases}$$

That is, data from sensors whose topics have an associated transformation rule is processed accordingly[7], while data from unknown sensors (or sensors without transformation rules) are managed by function $p_s(x)$.

In a traditional ETL tool there are two standard options for the choice of $p_s(x)$: either the null function $p_s(x) = \_$ corresponding to dropping data, or the identity $p_s(x) = x$ corresponding to upload raw values without transformations. In our exETL, instead, if no transformation rule is found, the process moves to the next layer, i.e. the $p_s(x)$ function has the effect of passing data x to the `Buffering layer` for a specific elaboration. In the `Buffering layer`, the sensor data x of each unknown topic t is aggregated into a temporary buffer $B[t]$ of m minutes (e.g., $m = 5$). After this time has elapsed, the content of buffer $B[t]$ is passed to next layer, namely the `ML pipeline layer`.

The `ML pipeline layer` is designed as a sequence of three main activities, building upon the methodological study of our previous research [3]. The first activity consists of extracting, from each buffer $B[t]$, a vector $\phi(t)$ of five features: the number of data objects observed in the m minutes slot, the minimum value, the maximum value, the average, and the standard deviation of values. The second activity takes these feature vectors and feeds them into a predefined set of binary classifiers; each binary classifier is pre-trained to detect if data corresponds to a specific topic among the known ones. Therefore, each classifier outputs the probability $p_i(t_j)$ of each sensor j to produce data of a specific known topic i. This activity produces a prediction map P, where the rows represent the classifiers, and the columns correspond to the sensors that each classifier was trained for, along with an additional column labeled *unknown*. The final activity takes the prediction map P and applies a maximum likelihood matching algorithm, whose output is a remapping of each unknown topic t as a new topic t'. The outcome will be either (a) the identification of additional sensor topics (that is, t' is found among those in T), and in this case new transformation rules are added for the semantic layer, or (b) the classification of the sensor as *unknown* (that is $t' \notin T$), and in this case data logging keeps going on. The detection process is in fact repeated over time intervals, on larger logs, possibly detecting more topics.

The ML pipeline needs for a pre-training phase ahead of deployment, conducted in an offline manner by transferring learning principles on historical datasets of possibly *other companies*. The detection phase, instead, is performed online on the live system. Another important issue is that the models embedded in the ML pipeline as classifiers need to be lightweight, with minimal computing requirements. In our case, after conducting a preliminary experimental study, described later in Section VI-B, we chose to use a decision tree as classifier.

For the more formal details concerning the mathematical models and their validation we refer the reader to [3].

C. Rule transformations

A key point in the implementation of exETL is the effective management of configuration files for transformation rules. To support the exETL configuration, we adopted an ontology

[7]In case of corrupted or badly formatted data, the rule application (during the semantic stage) throws an exception and the data and an error message are stored in an ERROR log file for offline scanning. For the sake of simplicity, this alternative path is not shown in Figure 3 and in the definition of h_s.

(shown in Fig.5 using the W3C Web Ontology Language – OWL) to formally describe the transformation rules. Fig.4 reports an extract of the XML-based configuration file containing the transformation rules associated with topics where sensor data is published. Each <topic> element represents a unique topic, linked to a sensor, and is associated with a transformation rule. For practical purposes, the transformation rule is identified by the element <transformationRuleID> that its an ID. Each topic has its own transformation rule, and these rules can be associated with multiple topics. Data from sensors comes in several formats. For most sensors, the data is in binary format because some textile machinery operates at high speeds, so the smAgent uses binary format and big-endian concatenation to minimize data conversions. This approach can indeed reduce latency and ensure efficient communication via MQTT. Data can also be a single value (an integer or a floating-point number) that needs to be converted into binary format, or a composite value made up of multiple elements, each of which is of a different type. In other cases, a more structured payload can be published as a JSON object.

Table I describes all transformation rules for the sensor data associated with the topics listed in Fig. 4. Each ID is linked to a specific transformation rule. In the exETL transformation process, the *strategy design pattern* is used and implemented in Python to select the transformation rule associated with each ID to execute. The input data is expected in the format associated with the ID, and it is transformed according to the rule assigned to that particular ID.

TABLE I
EXAMPLES OF TRANSFORMATION RULES FOR SENSORS' DATA ASSOCIATED WITH TOPICS

Rule ID	Data type	Char split	Input	Output
001	single value		binary	int
002	single value		binary	float(2)
003	single value		binary	float(4)
004	multi value	spacebar	binary	all int
005	multi value	spacebar	binary	all float(2)
006	multi value	spacebar	binary	all float(4)
007	key value		binary	json
008	key value		string	json

As an example, let us consider an industrial printer machine equipped with a sensor named inkStatus (as reported in Table II) that provides the percentage of ink remaining in the JSON format: {bl:50, r:75, g:85, b:80}, where bl is black, r is red, g is green, and b is blue. Being an advanced printer, it allows writing a JSON string to the PLC, so the smAgent reads this value and sends it to the broker in the Edge Node via MQTT. The transformation rule then applied is that with ID 008.

VI. EVALUATION

This section presents the results of exETL's evaluation. Specifically, we addressed the following research questions:

```
<?xml version="1.0" encoding="UTF-8" ?>
<topics>
    <topic
        name="EdgeNode/OLPT/machine_0/sensor_1"
        transformationRuleID="001" />
    <topic
        name="EdgeNode/OLPT/machine_0/sensor_2"
        transformationRuleID="001" />
    <topic
        name="EdgeNode/OLPT/machine_1/sensor_1"
        transformationRuleID="002" />
    <topic
        name="EdgeNode/OLPT/machine_2/sensor_1"
        transformationRuleID="003" />
...
    <topic
        name="EdgeNode/OLPT/printer/statusink"
         transformationRuleID="008" />
</topics>
```

Fig. 4. Example of configuration file with data transformation rules

```
<?xml version="1.0"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/
                1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
    <owl:Ontology rdf:about="topic-rules"/>
    <owl:Class rdf:about="#Topic">
        <rdfs:label>Topic</rdfs:label>
    </owl:Class>
    <owl:DatatypeProperty rdf:about="#name">
        <rdfs:domain rdf:resource="#Topic"/>
        <rdfs:range rdf:resource="xsd:string"/>
        <rdfs:label>name</rdfs:label>
    </owl:DatatypeProperty>
    <owl:DatatypeProperty
         rdf:about="#transformationRuleID">
        <rdfs:domain rdf:resource="#Topic"/>
        <rdfs:range rdf:resource="xsd:string"/>
        <rdfs:label>ruleID</rdfs:label>
    </owl:DatatypeProperty>
</rdf:RDF>
```

Fig. 5. Ontology for the data transformation rules

RQ1 How flexible is exETL compared to state-of-the-art alternative solutions?
RQ2 How effective is exETL in finding correct transformation rules for sensor streams whose topics are unknown?
RQ3 How efficient is exETL in terms of computing effort?

A replication package is available online [15] and contains the data sets and software artifacts used in the evaluation.

A. Design of the evaluation

To answer RQ1 we qualitatively compared exETL with two state-of-the-art and open-source software tools, namely Apache NiFi and Apache Kafka, that support real-time data management in distributed environments and therefore useful for implementing ETL processes. The key characteristics that we considered important in our context for such a comparison are described as follows.

Data transformation management relates to the capability to accommodate customization of the data transformation via domain-specific code. *Machine learning support* is a key aspect, especially when the ETL process (like in our case) has

to perform classification tasks. These tasks typically require the use of ML algorithms that learn how to assign a class label to concrete data from the problem domain. *Management interface* relates to the availability of a graphical web interface for managing and customizing data pipelines. *Use of external libraries* regards the possibility of using external libraries for complex data pipeline customizations and advanced ML algorithms. *Adaptability* can be defined as the ease with which the software tool may be adapted to the changing data processing requirements and contexts.

To answer RQ2 and RQ3, we conducted an experimental campaign, replicating the setting of a real installation in textile manufacturing. The experiments run on a workstation featuring an 1.80 GHz Intel(R) Core(TM) i7-8550U CPU with 16GB of RAM, Windows 10 Pro 22H2 as operating system, although the actual hardware is not relevant, being the computational resources strongly limited by the caps in the Virtual Machines (VMs). A simulation environment was created as detailed in the following. A VM in Oracle VirtualBox 7 was configured to replicate the actual edge node provided to the customers: it is a SBC with 2 virtual CPU cores capped at 50% execution and 4GB of RAM, running Ubuntu LTS 24.04. The computing capabilities are chosen to match those of a Raspberry Pi 4. Additional VMs were used to simulate the smart interfaces producing sensor data, reading test data from disk (as if they were obtained via modbus) and streaming them on a virtual network via MQTT. exETL together with a Mosquito MQTT broker were run on the simulated SBC as actual system processes. That is, a setting as close as possible to the architecture deployment shown in Figure 2 was built.

Following the guidelines of [3], a decision tree has been used as binary classifier in the ensemble of the ML pipeline layer, in its implementation in the library scikit-learn (version 1.5.1). The topic matching algorithm embeds a mathematical programming model which is solved by the OR-Tools Python library (version 9.11.4210).

To properly fine-tune the effectiveness of the classifier model, several preliminary experiments were run, comparing four types of supervised classifiers: Decision Tree, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Naive Bayes. Both binary and multilabel settings were checked. These preliminary results indicated decision trees to provide best tradeoff between accuracy and precision (exceeding 90% in nearly all tests) and computational times for prediction. Our checks always matched the results of [3]. The remaining blocks of the ML pipeline did not need fine tuning.

The experiments were carried out using collections of real production data from three clients in the high-fashion industry, with similar production settings. The distribution of sensor data occurrences per client are represented in Table II. The dataset of the third client was particularly rich. Data was grouped in 108 steps of 5-minutes each, resulting in simulations of 9 hours across 5 potentially unknown sensors. The tests were also repeated for durations of 7, 5 and 3 hours. For each time interval, tests were conducted to recognize 9, 5, 4, and 3 unknown sensors. Each test was repeated 10 times.

TABLE II
SIZE OF SENSOR DATA FROM ONE PRODUCTION YEAR

Topic	Client 1	Client 2	Client 3
airflowCount	249		646
airflowOff	255		514
vaporCount	713		24678
vaporOff	714		24025
vacuumCount	321		21298
vacuumOff	320		20736
spinCount		1291	
processPause			68
processStop			8204
operationPhase			1812
outputQuantity			2
inkStatus			104
vaporizerOff			4

TABLE III
ACCURACY OF ML PIPELINE LAYER

	Buff.(h)	Acc.
Best Setup of ML Pipeline layer	5	83%
	10	75%
	20	59%
Variant Without Matching Algorithm	5	49%
	10	50%
	20	67%

B. Results

RQ1: To answer this question we performed the qualitative comparison as designed in the previous subsection. The results of such a comparison are summarized in Table IV.

Regarding *Data transformation*, although NiFi, Kafka, Flink and Spark as non-custom solutions are designed for high-performance data flow/stream management, they allow to write code snippets only for simple data transformations using custom processors in programming languages (like Java, Groovy, and Jython in NiFi; C++, Go, or Python in Kafka; etc.). So these solutions are limited for complex data transformations, as they do not fully support code customization and require specific processors for such coding tasks. In contrast, exETL, being a custom solution written in Python with support from data science libraries like NumPy and Pandas, offers greater coding flexibility, enabling the implementation of complex transformations on large and heterogeneous data sets.

Concerning the aspect of *Support for Machine Learning*, exETL was purposefully designed around this feature, by embedding a specific ML pipeline using Python-based libraries. Spark provides a dedicated and scalable ML library (MLlib) for classification, regression, and clustering. Kafka, NiFi and Flink, instead, do not natively support any supervised classification technique for data pipelines that need to integrate a predictive model. Moreover, embedding ML models within

TABLE IV
COMPARISON OF TECHNOLOGIES FOR ETL PROCESSES: PYTHON, APACHE NIFI, AND APACHE KAFKA

	exETL (Python)	**Apache NiFi**	**Apache Kafka**	**Apache Spark**	**Apache Flink**
Type of tool	Solution based on a general purpose programming language	Tool for managing data flows	Data streaming platform	Big data processing engine for batch and streaming	Streaming-first data processing engine
Data transformation management	High flexibility with libraries (Pandas, NumPy)	Effective in managing continuous data flows	Limited, not designed for complex transformations	Supports parallel big data transformations with RDDs (Resilient Distributed Datasets) and DataFrames	Optimized for real-time streaming and supports batch transformations
Machine learning support	Full support for implementing ML techniques	Not natively designed for ML; can orchestrate flows and route data to external ML models	It does not directly handle ML; can route data to external ML platforms	Includes MLlib, a library for classification, regression, clustering, etc.	FlinkML supports basic ML algorithms, less developed than Spark
Management interface	Text-based (IDE, scripts)	Intuitive graphical interface	Text-based or via API	Text-based (scripts, notebooks)	Text-based or via web APIs
Use of external libraries	Wide range of libraries for any purpose	Integrated libraries for data flow	Libraries for integration with other platforms	Seamless integration with external ML libraries (e.g., TensorFlow, Scikit-learn)	It integrates external libraries; slightly less seamless than Spark
Adaptability	Highly adaptable for specific projects	Adaptable but less flexible for complex transformations	Adaptable but limited for data transformation	Adaptable for both batch and streaming processing but not suitable to implement Edge Node services	Adaptable for real-time streaming use cases but not suitable to implement Edge Node services

deployments of NiFi, Kafka or Flink at the edge would require specific integrations with external platforms (e.g., Kafka-ML and TensorFlow) and enhancements for *edge analytics* [16].

Regarding *Management interface*, Kafka, NiFi, Flink, and Spark have an intuitive web UI for managing ad customizing data pipelines. The current prototype of exETL has no UI yet. However, Python provides precise control via scripts and IDEs, benefiting developers working on ETL pipelines.

For the *Use of external libraries*, exETL is written entirely in Python, allowing the use of a wide range of libraries, including native C libraries, without platform dependencies. kafka and NiFy have become popular for building scalable and fault-tolerant data pipelines, but more complex data manipulation would require the implementation of external client programs using client libraries that are not always user friendly. Similarly for Spark and Flink.

Finally, concerning *adaptability*, in the considered textile manufacturing context and use case scenario where data transformation and dynamic classification are needed according to the acquisition of new machinery and equipment, a custom Python-based solution like exETL is maintainable over time, adapting to new requirements or technologies, and therefore it is a sustainable choice. In contrast, Kafka and NiFi are not well-suited for implementing complex data transformation and, therefore, less flexible for accommodating changes. Spark and Flink are resource cost-prohibitive for edge node services.

RQ2: As described earlier, the ML pipeline depicted in Fig. 3 is structured into three blocks: feature extraction, sensor classification, and the topic pattern matching. To assess and validate the predictive component within the ML pipeline layer, a series of preliminary tests were conducted on the selection of supervised classifiers, as outlined previously. Four datasets, derived from the data of three clients and detailed in Table II with 13 different topics, were employed for both training and testing phases on both binary and multilabel. These evaluations were part of a focused research effort aimed at enhancing system accuracy [3]. Results showed that the Decision Tree performed best, achieving an accuracy (in predicting single blocks) above 90%, with recall and F1 scores around 85% and a precision slightly exceeding 85%.

We finally compared two ML pipeline settings, by turning either on or off the topic pattern matching algorithm. That is, the test evaluates the efficacy of the last block in our ML pipeline, which is also the most computationally expensive. It was repeated for different buffer lengths, to understand the impact of such a parameter choice. Our results are reported in Table III. The use of the pattern matching block is crucial. Best results were obtained by using a buffer length of 5 hours, leading to an overall classification accuracy of 83%.

RQ3: To evaluate the computational effort of the ML pipeline layer, the tests were conducted using the data generated from combinations of 3, 4, 5, and 6 potentially unknown sensors data, aggregated into time intervals of 3, 5, 7, and 9 hours, with each configuration replicated 10 times, as explained above. The tests were performed directly on the virtual machine simulating the edge node SBC.

Table V reports average numerical results about computing times and CPU usage for each combination of test length and number of unknown topics. Computing times for each test are also shown by the jitter plot in Figure 6. It reports the length of tests on the X-axis, and the CPU time of the whole ML pipeline on the Y-axis. It can be noticed that the computational effort increases mildly as the number of buffering hours and sensors to be recognized increases. More in details, the ML pipeline layer maintains an average execution time of just almost over half a second for the test with 3 hours and 3 unknown sensors data, reaching about 2.5 seconds for the

TABLE V
EVALUATION OF EXETL PERFORMANCE BY HOURS AND NUMBER OF UNKNOWN TOPICS PROCESSED

Hours	Topics	Avg. CPU time	Avg. CPU usage
9	6	2.5677	99.30
9	5	2.3914	92.97
9	4	1.6845	99.50
9	3	1.2264	99.63
7	6	2.1131	98.89
7	5	1.7054	99.69
7	4	1.3990	99.05
7	3	1.0272	99.59
5	6	1.6588	99.45
5	5	1.3845	99.34
5	4	1.0798	99.84
5	3	0.8189	99.53
3	6	1.3553	89.56
3	5	1.0355	99.68
3	4	0.8291	99.03
3	3	0.6223	99.65

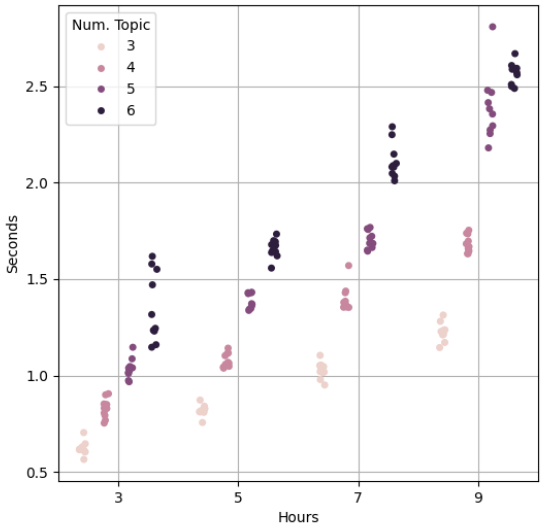

Fig. 6. Jitter Plot of seconds based on second

most demanding run with 9 hours and 6 unknown sensors. Concerning CPU usage we observe the following. In a realistic setting, when the detection process is idle, the CPU utilization of the whole ML pipeline layer is negligible (and in general the load on the edge node is very low). When the detection process starts (once every two hours), no I/O bottleneck is observed: the detection process is able to fully exploit the CPU (for very few seconds) and then goes back to idle.

We conclude that the ML pipeline computing effort is negligible in the economy of the overall architecture.

VII. THREATS TO VALIDITY

We are aware that internal, external and conclusion threats may affect the validity of our approach. To show its potential and weaknesses, we built a simulated environment, to be used for both fine tuning and for stress tests with real data sets. Our simulation setting faithfully virtualizes the production setting in terms of hardware and communication protocols. Nevertheless, as a possible threat to *external validity*, we mention that the extended ETL functionality for recognizing unknown sensors requires further experiments in industrial deployments to reach the production stage.

On our simulated environment we could measure several computational parameters, and compare several settings. Indeed, the ML pipeline consists of three stages (feature extraction, sensor classification, and topic matching). A threat to *internal validity* to exclude was obviously lack of accuracy and other effectiveness metrics during classification: our exETL showed excellent predictive power. Another threat to internal validity to exclude was the computational effort of the matching stage. Also in this case, our experiments allowed to understand that little computational effort is required, even on SBCs, scaling also well as the number of sensors (or the length of the observation buffers) increases. We also limited threats to *conclusion validity* by repeating the experiments multiple times; we made experiments varying buffer lengths, numbers of unknown topics, and observation time intervals, and making 10 repeats for each configuration.

VIII. CONCLUSION AND FUTURE WORK

We presented an on-premises data pipeline architecture for smart manufacturing, with a focus on its innovative ETL component. The dynamic rule management of exETL offers flexibility, enabling a smooth transition from legacy systems to Industry 4.0 standards. This is particularly impactful for manufacturers, especially in textiles, as it allows integration of machinery from different providers without needing software reconfiguration or data transfer adjustments. Additionally, we enhanced exETL with an ML pipeline that automatically identifies transformation rules for both known and new machinery.

Our research opens indeed to further challenges. One of them is to reduce the observation time needed to detect topics of unknown sensors, and thereby avoiding the loss of information when new machinery is attached live to the system. To this purpose, we postpone as future work to make the architecture self-adaptive so that the `Smart Interface` can dynamically create appropriate topics for unknown sensors that have been recognized and classified by exETL. Moreover, we want to investigate more on the exETL scalability capacity in handling a growing amount of unknown sensors (though rarely happens in a textile manufacturing system) and diverse operating conditions with edge resource constraints.

REFERENCES

[1] O. Oleghe and K. Salonitis, "A framework for designing data pipelines for manufacturing systems," *Procedia CIRP*, vol. 93, pp. 724–729, 2020, 53rd CIRP Conference on Manufacturing Systems 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827120305618

[2] M. Souibgui, F. Atigui, S. Zammali, S. Cherfi, and S. B. Yahia, "Data quality in etl process: A preliminary study," *Procedia Computer Science*, vol. 159, pp. 676–687, 2019.

[3] A. Ceselli, G. De Martino, and M. Premoli, "Identification of sensors in smart manufacturing via mutually exclusive multiple time series classification," *Journal of Intelligent Manufacturing*, pp. 1–17, 2024.

[4] M. Patel and D. B. Patel, "Progressive growth of etl tools: A literature review of past to equip future," in *Rising Threats in Expert Applications and Solutions*, V. S. Rathore, N. Dey, V. Piuri, R. Babo, Z. Polkowski, and J. M. R. S. Tavares, Eds. Singapore: Springer Singapore, 2021, pp. 389–398.

[5] ——, "Data Warehouse Modernization Using Document-Oriented ETL Framework for Real Time Analytics," in *Rising Threats in Expert Applications and Solutions*, V. S. Rathore, S. C. Sharma, J. M. R. Tavares, C. Moreira, and B. Surendiran, Eds. Singapore: Springer Nature Singapore, 2022, pp. 33–41.

[6] C. Boulahia, H. Behja, M. R. Chbihi Louhdi, and Z. Boulahia, "The multi-criteria evaluation of research efforts based on etl software: from business intelligence approach to big data and semantic approaches," *Evolutionary Intelligence*, pp. 1–26, 2024.

[7] E. Y. Nakagawa, P. O. Antonino, F. Schnicke, R. Capilla, T. Kuhn, and P. Liggesmeyer, "Industry 4.0 reference architectures: State of the art and future trends," *Computers & Industrial Engineering*, vol. 156, p. 107241, 2021.

[8] J. Kaiser, D. McFarlane, G. Hawkridge, P. André, and P. Leitão, "A review of reference architectures for digital manufacturing: Classification, applicability and open issues," *Computers in Industry*, vol. 149, p. 103923, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361523000738

[9] T. Sauter, "Integration aspects in automation - a technology survey," in *2005 IEEE Conference on Emerging Technologies and Factory Automation*, vol. 2, 2005, pp. 9 pp.–263.

[10] ——, "The continuing evolution of integration in manufacturing automation," *IEEE Industrial Electronics Magazine*, vol. 1, no. 1, pp. 10–19, 2007.

[11] A. Ismail, H.-L. Truong, and W. Kastner, "Manufacturing process data analysis pipelines: a requirements analysis and survey," *Journal of Big Data*, vol. 6, no. 1, pp. 1–26, 2019.

[12] D. S. Milojicic, "The edge-to-cloud continuum," *Computer*, vol. 53, no. 11, pp. 16–25, 2020. [Online]. Available: https://doi.org/10.1109/MC.2020.3007297

[13] L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, "A unified model for the mobile-edge-cloud continuum," *ACM Trans. Internet Technol.*, vol. 19, no. 2, apr 2019. [Online]. Available: https://doi.org/10.1145/3226644

[14] S. Bergamaschi, F. Guerra, M. Orsini, C. Sartori, and M. Vincini, "A semantic approach to etl technologies," *Data & Knowledge Engineering*, vol. 70, no. 8, pp. 717–731, 2011, pushing Artificial Intelligence in Database and Data Warehouse Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169023X11000309

[15] A. Ceselli, "Replication Data for: Non-invasive software architecture for data pipelines with legacy support in smart manufacturing," 2025. [Online]. Available: https://doi.org/10.13130/RD_UNIMI/OHGUTH

[16] S. Nayak, R. Patgiri, L. Waikhom, and A. Ahmed, "A review on edge analytics: Issues, challenges, opportunities, promises, future directions, and applications," *Digital Communications and Networks*, vol. 10, no. 3, pp. 783–804, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864822002255