Adaptive Architectures: Designing an Industry 4.0 Prototype Based on Scientific Research

Joachim R Baumann*, Phillip Nielsen*, Sebastian Revsbech Christensen*,
Simone Cosentino*, Oskar Præstholm*, Hans Askov*

University of Southern Denmark, SDU Software Engineering, Odense, Denmark
Email: * {jobau19,phnie19,sechr17,sicos23,ospra20, haask19}@student.sdu.dk

***Abstract*—This paper explores an Industry 4.0 (I4.0) ball pen production system, investigating the feasibility of existing technologies to meet key quality attributes (QAs): interoperability, availability, deployability, and scalability. Emphasizing system scalability, the study designs a prototype software solution inspired by recent scientific articles. The prototype, leveraging MQTT and Kafka brokers, achieves a balance between efficiency and robustness. By centralizing data through Kafka and utilizing MQTT for lightweight IoT devices, the system handles over 20 sensors, maintaining latency below 1 second. Promising for I4.0 production, the paper recommends future work to test additional quality attributes and system components.**

***Index Terms*—Ball pen production, Industry 4.0 production, I4.0 production, Quality attribute experiment, Cell based production**

I. Introduction and Motivation

While older software-systems have established an industry standard for their architectural design patterns, the growing requirements for new systems make it increasingly harder to keep up. In order to adapt, scientific authors examine the different architectures and their impact on quality attributes. These scientific articles serve as an inspiration for system developers for their system designs. For new production environments like Industry 4.0 (I4.0) the adaptation of a standard architectural design patterns is ongoing. This study uses scientific articles to design a prototype software-solution for an industry 4.0 production. The research resulted in a prototype using a combination of MQTT and Kafka.

II. Problem, research questions and Approach

A ball pen manufacturer needs to create a fully automated production line, with the specified requirements of high availability and easy interoperability, to avoid unnecessary downtime. The company wants the software architecture to support potential growth in production, so the ability to add more cells must be supported. When a fault occurs within the system architecture it needs to respond automatically to maintain high availability. To facilitate early problem fixing, the system should be monitored, either by a separate system or internally. All the data collected through monitoring needs to be sent to a separate system, to be analyzed in order to optimize the production.

1) Which architectural patterns would best align with the system's quality attributes?
2) Which tech stack could contribute positively towards the required quality attributes?
3) Which framework should manage internal communication between subsystems?

The following steps were taken to answer this paper's research questions:

1) Analyse the problem, and set a scope
2) Find related works and articles
3) Review chosen articles
4) Analyse article solutions
5) Use cases and Quality attribute scenarios
6) Design solution
7) Prototyping and testing

III. Related work

This Section addresses existing contributions by examining similar experiments conducted, and existing standards in the I4.0 domain. In total, 4 papers are investigated.

Article [1] summarizes the evolution into the industry 4.0 system, and describes the fundamental concepts and goals for I4.0 systems. By localizing responsibilities throughout the system, it becomes possible to adapt faster, enabling quicker decision-making. Encapsulating responsibilities to individual parts of the system, increases overall system interoperability and resource efficiency which allows the choice of different software for individual system parts. In principle, this should also increase modifiability and deployability, since each part of the system works independently and any updates or changes would only affect the singular part.

By following these industry 4.0 principles and concepts, integration becomes possible and the production can make use of value-creation networks, which should allow for considerable optimization of processes and possibly gain insight into new aspects of revenue. With a dynamic production network, data can be collected throughout the whole product life cycle.

In [2] a state-of-the-art production system developed for I4.0 systems has been proposed. The production system in the study focuses particularly on flexibility and adaptability

in the production system and proposes a model to fulfill these quality attributes. The production system is built with a Microservice architecture, as the different subsystems are abstracted away by being dockerized. This causes the system to be adaptable and flexible, as small changes to the micro-services, in case the requirements change, can easily be deployed. The subsystems then communicate through a message bus. The production line itself is built with a four-cell system, controlled by PLCs, that can function in multiple complex processes.

In [3] a pilot study of the interoperability quality attribute in a minimum viable product (MVP) of producing a drone has been analyzed. The setting of creating the drone is a modern state-of-the-art I4.0 laboratory, where multiple communication protocols and other assets, e.g. the cells in the production line, are required to work together to produce the product. Through an analysis of these assets communicating, it is concluded that the interoperability in I4.0 systems is still immature. This is because of a lack of various requirements of interoperability from the different assets, such as a lack of good documentation, missing external interfaces, and other communication technologies. The reason for this could be an apparent lack of understanding of the necessity of asset interoperability in an I4.0 context.

For insight into how an I4.0 system could be applied to actual production, article [4] analyzes several other scientific articles on the topic. The article concludes with the creation of a prototype, with the purpose of testing the system architecture. The architecture for the prototype is based on a comprehensive literature review and gap analysis of industry 4.0 platforms.

As seen in figure 1 from article [4] the majority of the literature related to I4.0 uses different architectures, and an industry standard has not been formed. The article concludes that diverse quality attribute prioritization drives the different system architectures in different directions. After reviewing several of the different architectures, the authors decided that their system prototype's requirements aligned with either a micro-services or event-based architecture. In the end, when comparing the two, the event-based system allowed for higher performance, while keeping the interoperability aspect. The micro-service architecture added additional layers of complexity and fell behind in performance. In the end, the event-based architecture was decided for the prototype, as it had many of the perks of a micro-services architecture, while still maintaining a high performance.

IV. Use Case and Quality Attribute Scenario

This Section introduces the use case and the specified four QASes. The QASes are developed based on the use case.

To facilitate an analysis of the needs of the ball pen manufacturer that is facing potential growth in their production, a use case analysis of the required system has been performed. The assumptions for creating the use cases are seen in table I.

ID	Assumption	Reasoning
A1	The pen consists of 5 parts	The disassembled pen shows this to be the case
A2	4 assembly steps are conducted by 4 individual cells	Minimum number of steps required to assemble the 5 parts of the pen.
A3	Each cell contains sensors to monitor its own health and progress in its process.	Sensor and monitor data need to be collected in order to operate the system.
A4	Each cell has a box of parts that is fed into the assembly line to combine with the pen	The different cells need access to their respective parts.
A5	Each cell monitors execution time, availability, performance, quality, and OEE	Data is needed to run diagnostics and perform optimization of the process.
A6	Different cells may require software written in different languages.	For optimization and diagnostics.
A7	The assembly line has to run without operator interaction (fully automated)	To reduce production times and labor costs.
A8	A database should be used.	To store relevant data for logs.
A9	The assembly line contains both parallel and sequential operations.	Some parts can be combined simultaneously, while the last part is dependent on two other cells.
A10	The system uses pre-made parts therefore only assembly of the ball-pen is required.	The parts of the pen to be assembled are already made in a different area of production.
A11	Quality control happens at each cell.	The sensors at the cells are able to conduct sufficient quality control.

TABLE I: Table of Assumptions and Reasoning

These assumptions were the basis for the use cases that are shown as in the following subsection IV-A.

A. Use case

Table II describes the process of starting the production system. The primary actor is the production manager who is responsible for initiating the process. The precondition of this is that he is logged into the software and that all the necessary materials, machine setups, and preliminary checks are ready. The main flow describes the production manager starting the production system and the subsequent steps this entails. The postcondition is that the system is operational. An alternative flow describes when the system diagnoses an issue.

Table III is the use case for the process that happens to facilitate the error diagnostics of the system. The primary actor for this is again the production manager. The preconditions describe the conditions for the error diagnosis to take place. The main flow describes the steps the production manager must take to achieve this. The post-conditions indicate that the result of this is that an alarm has been activated, signaling the error. The alternative describes the case where the cells respond with a bad diagnosis, and the system can't handle it automatically.

Table IV outlines the criteria for halting production and explains the subsequent actions that occur once the production manager has initiated the "stop production" command in the

Platform name	Industry sectors supported	Architecture style/type	Communication	Knowledge protection and sharing mechanisms	Platform service extensibility approach	Implementation technologies
NIMBLE (Innerbichler et al., 2017)	Yes	Microservice infrastructure	Service-oriented	Data sharing service and product ontology	Hardcoded	Web interface, cloud-based microservice
IDARTS (Peres et al., 2018)	Yes	General layered architecture		Knowledge Management Component; data workflow	N/A	Java Agent DEvelopment framework (JADE)
IoT energy platform (Terroso-Saenz et al., 2019)	No	General layered architecture		UML class-based information model; data workflow	Hardcoded	Nimbits
RTMIIS (Zhang et al., 2015)	Yes	SOA	Service-oriented	N/A	Hardcoded	Cloud-based micro-service
Government affairs service platform (Lv et al., 2018)	No	SOA	Service-oriented	User service	Hardcoded	Web VRGIS engine, cloud-based microservice
A collaborative knowledge transfer platform (Cotrino et al., 2021)	No	General layered architecture	RESTful API	Database	Hardcoded	PHP, JavaScript, MySQL
Middleware for Intelligent Automation Platform (Coito et al., 2020)	Yes	Specialized middleware in general layered architecture	OPC UA Standard over Time Sensitive Networks	Cloud service and database warehousing	Hardcoded	Cloud-based microservice
Fog-of-Things platform (Verba et al., 2019)	No	Platform-as-a-Service	Message routing	Cloud service	Hardcoded	Cloud-based microservice
Automated flow-shop manufacturing system (Liu et al., 2019)	Yes	General layered architecture	OPC UA with database	Multi-view synchronization	Hardcoded	J2EE and Unity3D
SCM system with a blockchain-enabled architecture (Wang et al., 2020)	Yes	General layered architecture	Peer-to-peer blockchain network	Blockchain database; Traceability and visibility service	N/A	Validated by experts in SCM
Simulation Platform for Virtual Manufacturing Systems (Dobrescu et al., 2019)	Yes	SOA	Service-oriented	Cloud service and data store model	Service-oriented integration	Cloud-based microservice
DIGICOR (this paper)	Yes	Specialized EDSOA	Event-based	Authentication service and ontology layered architecture	App-store based	Angular 4 front-end, cloud-based back-end, Kubernetes

Fig. 1: Overview of system-architectures used for I4.0

Use Case: Start Production	
ID:	U01
Primary Actor:	Production manager
Secondary Actor:	None
Short Description:	This process is responsible for the start of the production and the workflow connected to it.
Preconditions:	• The production manager, responsible for initiating production cycles, is logged into the software. • All necessary materials, machine setups, and preliminary checks for production cells have been completed and are ready for operation.
Main Flow:	1) The production manager initiates the production via the central control software by clicking "start production" 2) The production orchestrator conducts a status check for all cells, confirming the readiness for production. 3) Upon successful status confirmation, each cell sends a ready signal back to the orchestrator. 4) The control software notifies the production manager that the production has started. 5) The user interface switches to display the production real-time status information screen.
Post conditions:	• All production cells are operational, and the production manager is monitoring the system through a user interface.
Alternative Flow:	• If the cells respond with a bad diagnostic, the production manager is notified, and the production start is canceled.

TABLE II: Use Case Specification for Starting Production

Use Case: Error diagnosis	
ID:	U02
Primary Actor:	Production manager
Secondary Actor:	None
Short Description:	This use case describes what happens to the system when an error has occurred. The production manager is logged into the system.
Preconditions:	• The production manager is logged into the centralized control system. • An error has been detected in the production line, halting the production process.
Main Flow:	1) The production manager clicks "Error diagnosis" 2) The production orchestrator checks status for each cell in the production. 3) Each cell sends diagnostic information in return. 4) The diagnostic data is displayed at UI. 5) An alarm or notification begins.
Post conditions:	An alarm has occurred.
Alternative Flow:	• If any of the cells respond with a bad diagnostic, the production manager is notified. • The system triggers a diagnostic routine in order to identify the issue. • If the issue cannot be handled automatically, the production is stopped until manual intervention is initiated.

TABLE III: Use Case Specification for Error Diagnosis

orchestration software's user interface, which controls the system.

B. Quality attribute scenarios

Based on the use cases, four quality attributes (QA) are identified to achieve the system.

1) Interoperability
2) Availability
3) Deployability
4) Scalability

Interoperability is important, as the multiple systems for production have to work together. The use cases show that the system contains various software and hardware components that need to work in unison. Interoperability ensures that this can happen. Data flow is also important, as seen in table II where the organizational system requires diagnostics of the cells, ensuring that they are operational to show the production manager that everything is operational in the user interface. The same is the case for III where error diagnostics of the cells require interoperability between the different systems to identify the issue. Interoperability is also part of ensuring scalability, as it allows for the integration of new technologies without interrupting the entire workflow.

Availability is important for a system such as this, as any downtime directly translates into lost productivity. It is

Use Case: Stop Production	
ID:	U01
Primary Actor:	Production manager
Secondary Actor:	None
Short Description:	This process is responsible for stopping the production and the workflow connected to it.
Preconditions:	• The product manager is logged into the software. • The production is running.
Main Flow:	1) The production manager clicks "Stop Production" from the orchestration software user interface. 2) The production orchestrator sends a stop production message to each cell. 3) The software notifies the production manager that the production has been stopped.
Post conditions:	The production has stopped and the production manager has confirmed the pause in operations.
Alternative Flow:	Not applicable

TABLE IV: Use Case Specification for Stopping Production

also important as seen in the process flows in the table, where the production manager has to make informed decisions based on the feedback from the user interface in real-time. If any parts of the system are unavailable, this might be impossible. It is also directly tied to maintenance and upkeep, as these are also based on system feedback as seen in table III. If this is not available, this can have consequences for the equipment or the safety of the workers.

Deployability allows the system to be easy to set up quickly and configure with minimal effort. Depending on the system, different deployability orchestrators may have an advantage over others. Deployability also denotes the system's recovery speed, ensuring that in case an issue is identified the system will quickly be able to recover, minimizing downtime. This is also a requirement for the system to gain maintenance, as the production demand grows the system can be scaled to accommodate the increased load.

Scalability is important since the ball pen manufacturer is facing potential growth. A system with scalability will allow the manufacturer to scale up without any downtime in production. The system will also be able to handle the potential increase in error checks that have to be performed.

V. The solution

This section describes the design of a prototype, which will be used to test if the selected architecture supports the quality attributes.

The initial candidates for the architecture, based on the research articles from chapter III, pointed towards implementing the system in a micro-service pattern or an event-driven pattern. The conclusion was to use a micro-service pattern for overall communication between each system and within the micro-service pattern, to implement the production system with the event-driven software architecture. By using an event-driven architecture for the conveyor belt and its sensors, the system becomes capable of sharing real-time data, while adapting to changes if necessary. via asynchronous communication. By using a message bus for communication, the individual parts can remain loosely coupled increasing the overall interoperability of the system, while the different system components, become individually scalable. The downside to using these architectural patterns is the increased complexity in implementation. While the architecture behind the conveyor belt does not require a lot of different components, if the production later needs to add additional features, the complexity could become problematic.

The entire system consists of many individual services, ranging from log collection to an inventory management system. Each system contains several sub-systems with different purposes see figure 2.

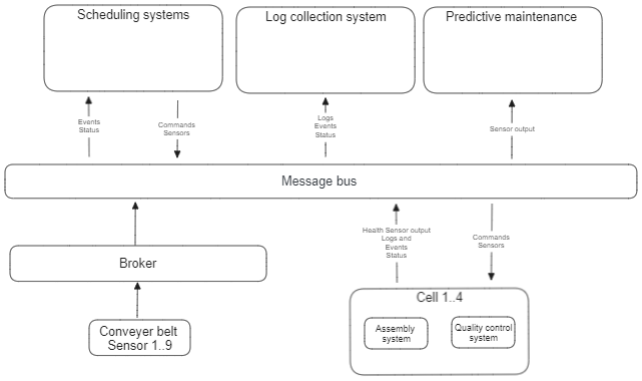

Fig. 2: The physical production layout

Designing the architecture as a whole and making design decisions for each system, would be too time-consuming and out of scope, considering the purpose is to create a testable prototype for an I4.0 production. As a result, the proposed solution will solely focus on the production system.

A. Designing the production

The physical production consists of 4 cells, each adding 1 additional part to the pen. Between these cells, there are sensors, which tell the next cell when the pen is ready. An illustration can be seen in figure 3. Cells one and two operate in parallel while three and four operate sequentially as defined in [2].

Each cell contains a robotic system responsible for assembling or combining parts to create the pen. The cells contain several sensors ranging from when the pen is ready for further assembling to collecting health data, from the cell. By having sensors collecting health data frequently, it should become possible to predict maintenance and investigate why errors occur.

1) Message Bus: To manage this large quantity of data, the event-driven architecture becomes relevant. By implementing a message bus to manage the data flow within the system, each cell keeps its interoperability and stays individually scalable. The message bus is responsible for receiving all the data from the different sensors and creating topics, where the relevant

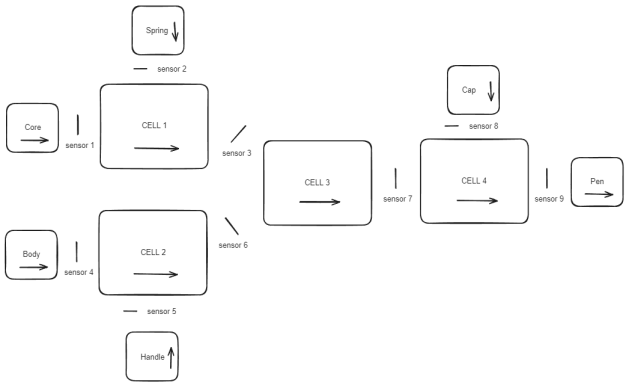

Fig. 3: The physical production layout

recipient can subscribe to receive the data. For the production system, several MOMs (Message Oriented Middleware) were considered. According to [5] the 4 most prominent within production systems are: MQTT, AMQP, and Kafka, which are all based on a broker, and ZeroMQ, which is brokerless.
Kafka has an edge over the others, due to the ability to replay messages: "A unique approach having durable saved message streams and the ability to jump back and forth is taken by Kafka" [5]. Kafka does this better than the other communication patterns while also allowing multiple publishers to write on the same topic.

Kafka is well-suited for real-time data streaming and event-driven architectures, while MQTT is tailored for lightweight, efficient communication in IoT scenarios. For that reason, each sensor between the cells will use the MQTT message bus to keep it lightweight and limit hardware usage. Kafka will mainly be responsible for handling the enterprise part of the system. However, a bridge/proxy between MQTT and Kafka will be created to analyze and save the data from the cells. Using both Kafka and MQTT collaboratively within the systems, allows the architecture to reap the benefits of both keeping the sensors lightweight, and having the Kafka capabilities for the rest of the system. Working collaboratively with Kafka and MQTT has been implemented successfully before as seen in [6]. While the sensors between cells will be communicating using the MQTT bridge, each individual cell will be transmitting its data directly to Kafka.

2) Database Choice: The production management system itself, will not have its own database, but the log data from the sensors still needs to be stored. This responsibility lies within the log collection system, which collects the data from Kafka via. an internal API. The data is then stored within a time series [7] database for later or ongoing analysis. By storing this data the system becomes able to analyze itself in various ways. By continually checking the data sent from the sensors, it becomes possible to create a predictive model, which makes fault prevention possible. In the case a fault occurs, it should be detected quickly by monitoring the heartbeat continuously, through the data sent.

VI. Evaluation

This Section describes the evaluation of the proposed design. Section VI-A introduces the design of the experiment to evaluate the system. Section VI-B identifies the measurements in the system for the experiment. Section VI-C describes the pilot test used to compute the number of replications in the actual evaluation. Section VI-D presents the analysis of the results from the experiment.

A. Experiment design

This chapter describes in detail the testing done on the prototype designed in chapter V, to validate its ability to fulfil the quality attributes.
This experiment is supposed to test the scalability quality attribute associated with the system architecture. To test this a prototype, mainly consisting of mocked sensor data, transmits the mocked data to Kafka through a MQTT broker. The Kafka message bus creates a topic, to which a subscriber created in Spring subscribes. The subscriber transmits the data to a Postgres database, where it is stored for analysis. All of this is managed by a docker-compose network. By using docker-compose for the different services it becomes possible to scale based on demand, which is what this experiment is testing. The purpose of the experiment is to test if Kafka can handle increased traffic load from the publisher, which sends the data. To achieve this, several publisher services are booted up at the same time. By testing, if Kafka can handle additional stress from the publishers, we gain insight into the system's ability to scale, if the company wants to add additional cells and therefore more sensors to the production.
To determine if the architecture meets the requirements set for the quality attribute a hypothesis has been formed:
"The architecture will be able to maintain a transmission time below 2 seconds while transmitting data from up to 20 sensors"

B. Measurements

The command "docker-compose up" is used to initiate each test. After having the system run for 4 minutes, the publisher services are manually closed through Docker desktop. Within the subscriber-service a REST API has been created, which exports the data from the Postgres database to JSON format. After formatting the data to csv, it is imported to Excel ready for analysis. The part of the data of interest is the timestamp the publisher sends, which indicates when the sensor has picked up the data, and the timestamp the Postgres database creates once it lands in the database. By subtracting the publisher timestamp from the database timestamp, we're able to calculate the ms it takes to transmit the data from the "sensor/publisher" to the database.

C. Experiment Data

After running the experiments accessing the endpoint that creates a JSON file, and converting that JSON file to csv format the data was assembled into a single csv file. Figure 4 shows a data example of this.

4_sensors	8_sensors	12_sensors	16_sensors	20_sensors
00.00.00	00.00.00	00.00.01	00.00.01	00.00.01
00.00.00	00.00.00	00.00.01	00.00.00	00.00.01
00.00.00	00.00.00	00.00.00	00.00.00	00.00.01
00.00.01	00.00.01	00.00.01	00.00.01	00.00.00
00.00.00	00.00.00	00.00.01	00.00.01	00.00.00
00.00.00	00.00.00	00.00.01	00.00.00	00.00.00
00.00.00	00.00.00	00.00.01	00.00.00	00.00.01
00.00.01	00.00.01	00.00.00	00.00.00	00.00.01

Fig. 4: Small data sample

The data is stored as a time object, which goes dd(days)-hh(hours)–ss(seconds). In the example on 4 it can be seen that the largest transmission times hover around 1000 ms.

D. Analysis

To gain an insight into all the data, each sensor's data was plotted to illustrate the differences and create a comparison.

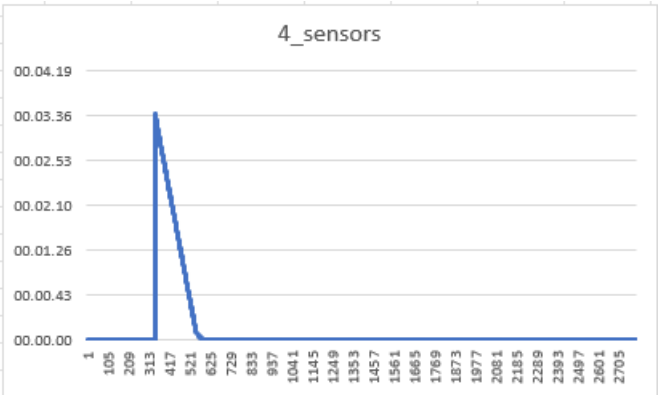

Fig. 5: 4 sensors plotted

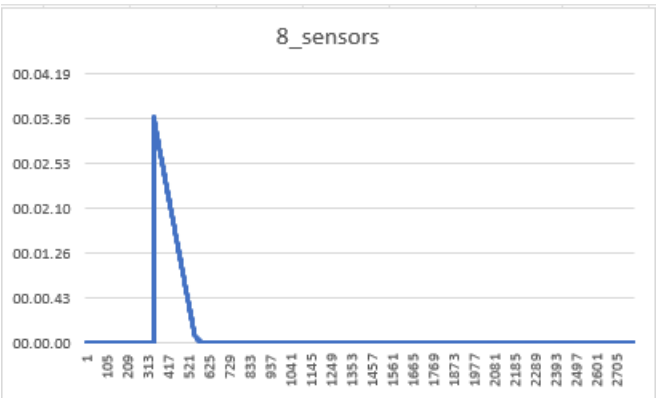

Fig. 6: 8 sensors plotted

We can see on the graphs that sensors 4 (figure 5) and 8 (figure 6) tend to stay at sub 1000 ms transmission type, with similar outliers. When inspecting these outliers manually it can be seen that it is the first data sent to Kafka, which lingers

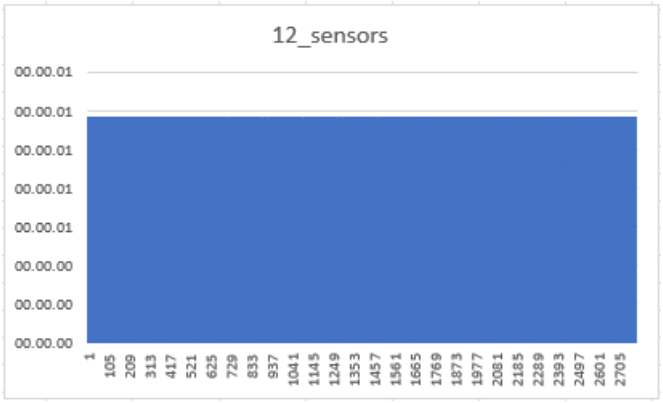

Fig. 7: 12 sensors plotted

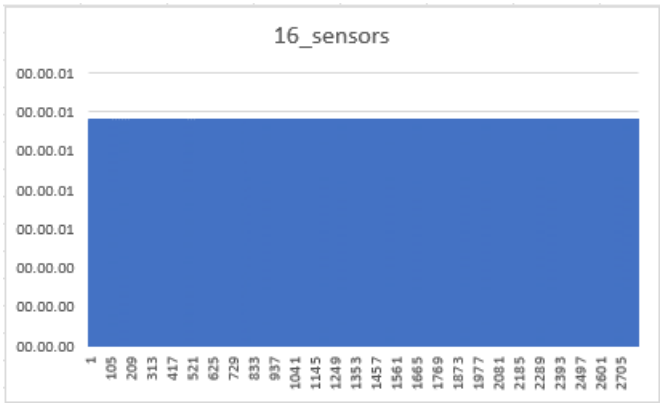

Fig. 8: 16 sensors plotted

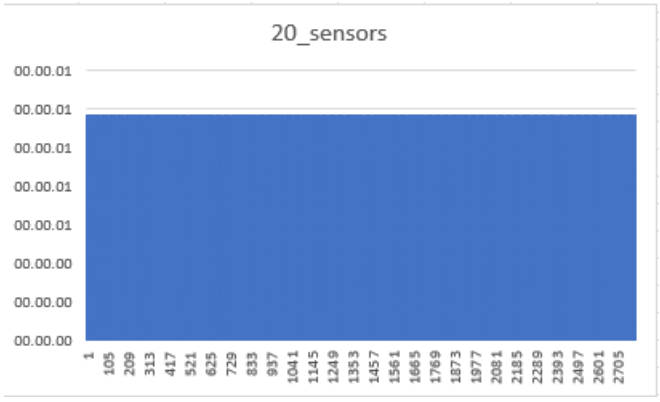

Fig. 9: 20 sensors plotted

until Kafka has established a connection to the database. Once the connection is established there is already live data flowing through the system, so Kafka stores the data, and sends it once the live data stream either pauses or is stopped. This results in a few outliers, matching up with the lifetime of the system during the experiment. Once the test has 12+ sensors, the amount of data sent through the system increases, because of this, and to keep the data comparable the data was limited to 3000 samples from each test. By removing data from the past 3000 samples, the data that would have had an extended

lifetime in the system from the experiments with 12, 16, and 20, does not display in the graphs. By looking past the outliers it can be concluded that the data stays below 1s response time when scaled from 4 to 20 sensors. This means the system architecture allows scaling of producers within the system, and therefore by extension allows the addition of several cells to the system and confirms the hypothesis.

VII. Future work

With the current prototype setup, the only load on Kafka is through the sensors, mocked with the producer service. To reach a state where it can be confidently concluded that the software architecture is viable to support the production of the I4.0 standard, further parts of the system would have to be mocked. By mocking more systems it would be easier to tell if Kafka could maintain a low ms on the data transmissions while supporting a complete production architecture. Other quality attributes than scalability should also be tested, more specifically the availability and deployability attributes. With the current prototype, the system is handled within docker-compose, which has sole responsibility for the deployability. It could be very interesting to compare the capabilities of docker-compose and an alternative like Kubernetes. While testing the scalability of Kafka, the system had an increasingly higher load each time it was booted up. To test the availability it would be a good idea to do live load testing and stress testing, where the system gets gradual increases or even major spikes in data traffic to see the response.

VIII. Conclusion

In conclusion, our exploration into the ball pen production system has made strides toward a more intelligent and efficient manufacturing approach. Drawing inspiration from key literature, especially [4], we crafted a prototype that blends Kafka and MQTT technologies. This strategic combination ensures our system functions smoothly with small sensors while offering scalability to handle increased data demands. The pivotal scalability test gauged the system's capability to manage heightened workloads with the addition of more production elements. Encouragingly, our prototype demonstrated the ability to maintain data transmission times below 1000 milliseconds, even with the integration of twenty sensors. This success not only affirms the system's adaptability for accommodating additional production cells but also positions us for future developments. Nevertheless, it's crucial to acknowledge that our current prototype, while showing promise in scalability, remains in its early stages. Further refinements are necessary, including simulating additional system components, scrutinizing attributes related to availability and deployability, and conducting more in-depth testing under realistic workloads. Our endeavor transcends academic pursuits, aiming to contribute to the evolution of manufacturing processes by infusing intelligence and adaptability. This project serves as a foundational step for ongoing exploration and advancements in manufacturing. Envisioning a future where manufacturing processes are not just intelligent but continually improving, we anticipate the unfolding paradigm shift in adaptable and efficient production systems aligned with Industry 4.0. In summary, the presented prototype and its evaluation substantiate the viability of the proposed system architecture, representing a key advancement toward intelligent and scalable manufacturing processes. This work provides insights into the future landscape of manufacturing, fostering a pathway for the integration of smart and dynamic production systems that align with the requirements of Industry 4.0. [4]

REFERENCES

[1] H. Lasi, P. Fettke, H. G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business and Information Systems Engineering*, vol. 6, pp. 239–242, 8 2014.

[2] S. C. Jepsen, T. Worm, A. Johansen, S. Lazarova-Molnar, M. B. Kjærgaard, E.-Y. Kang, J. Friederich, J. E. H. Mena, R. Soltani, S. L. Sørensen, and J. H. Schwee, "A research setup demonstrating flexible industry 4.0 production," in *Proceedings of the International Symposium Electronics in Marine (Elmar) 2021*, September 2021, pp. 143–150.

[3] S. C. Jepsen, T. I. Mørk, J. Hviid, and T. Worm, "A pilot study of industry 4.0 asset interoperability challenges in an industry 4.0 laboratory," in *IEEE*, 2020, pp. 571–575.

[4] Z. Liu, P. Sampaio, G. Pishchulov, N. Mehandjiev, S. Cisneros-Cabrera, A. Schirrmann, F. Jiru, and N. Bnouhanna, "The architectural design and implementation of a digital platform for industry 4.0 sme collaboration," *Computers in Industry*, vol. 138, 6 2022.

[5] T. U. München, I. Robotics, A. Society, I. of Electrical, and electronics Engineers, *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE) : 20-24 Aug. 2018.*

[6] Mqtt vs. kafka: Friends, not foes, in the world of real-time iot data processing. [Online]. Available: https://www.hivemq.com/blog/mqtt-vs-kafka-real-time-bidirectional-data-processing/

[7] What is a time series database? how it works use cases — hazelcast. [Online]. Available: https://hazelcast.com/glossary/time-series-database/