Architecture Blueprint Enabling Distributed Digital Twins

Frank Schnicke
Fraunhofer IESE
frank.schnicke@iese.fraunhofer.de

Daniel Espen
Fraunhofer IESE
daniel.espen@iese.fraunhofer.de

Pablo Oliveira Antonino
Fraunhofer IESE
pablo.antonino@iese.fraunhofer.de

Thomas Kuhn
Fraunhofer IESE
thomas.kuhn@iese.fraunhofer.de

ABSTRACT

Mass production today is optimized for large lot sizes, and changes to industrial production lines are effort-intense, time-consuming, and costly. The fourth industrial revolution, Industry 4.0 (I4.0), aims at reducing the effort needed for changes in industrial production lines. The key benefits of next-generation manufacturing systems are less downtimes and the production of small lot sizes down to lot size 1. I4.0 does not introduce a silver bullet technology, but requires a transformation of the system architecture of production systems. In the literature, however, there systematic guidance for designing manufacturing systems that address central I4.0 use cases like plug'n'produce and end-to-end communication is still missing, as are details on the infrastructure needed to enable I4.0 technologies such as Digital Twins. To contribute to filling this gap, this paper presents (i) a Digital Twin architecture blueprint driven by central I4.0 use cases and (ii) a prototypical open-source implementation of the architecture using the concept of the Asset Administration Shell.

CCS CONCEPTS

• **Computer systems organization**; • **Architectures**; • **Distributed Architectures**; • **n-tier architectures**;

KEYWORDS

Plug'n'Produce, Industry 4.0, Industrial Internet of Things, Digital Twin, Asset Administration Shell, Software Architecture

ACM Reference Format:
Frank Schnicke, Daniel Espen, Pablo Oliveira Antonino, and Thomas Kuhn. 2021. Architecture Blueprint Enabling Distributed Digital Twins. In *7th Conference on the Engineering of Computer Based Systems (ECBS 2021), May 26, 27, 2021, Novi Sad, Serbia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459960.3459978

1 INTRODUCTION

Industry 4.0 (I4.0) is the fourth industrial revolution [1]. As in the previous revolutions, I4.0 is triggered by the availability of a new technology that enables groundbreaking changes to production processes [2]. I4.0 is the end-to-end digitalization of manufacturing processes enabled by the availability of fast and reliable IT technology [2]. Similar to previous industrial revolutions, it will significantly increase the efficiency of mass production and decrease unit cost. It will also improve connectivity, enable new digital business models, and enable rapid re-configuration of manufacturing lines for the production of new product types [3].

A major idea of I4.0 is to change the architecture of a plant by replacing the hierarchical layers of the automation pyramid [4] to enable peer-to-peer communication across layers, which leads to the Industrial Internet of Things (IIoT). I4.0 aims to optimize processes and enable automation of steps previously executed manually through architectural and conceptual changes. One of these steps is the commissioning of new plants. Today, plant setup and configuration consumes a considerable amount of time. For instance, setting up a single device can take up to 90 minutes [5]. This effort is driven mainly by the need for human intervention. Thus, increasing the automation of commissioning steps can drastically reduce overall plant operation costs. Additionally, any changes of manufacturing processes, e.g., the introduction of a new device, create high costs. This is because every change requires a change in the automation software, and all of these changes must be tested on the manufacturing line, which typically requires a full stop of the production until all changes are done. Again, faster device integration will lead to a significant reduction in costs. plug'n'produce promises to introduce plug'n'playability to plants and leverage it to reduce human effort needed for the integration of new devices.

Many use cases related to I4.0, including end-to-end communication and integration of IT and ERP systems, greatly benefit from Digital Twins (DT) [6]. However, existing research mainly focuses on the application level of DTs, i.e., on how to integrate DTs with applications [7] and what data should be provided by DTs for use cases like service-based production [8]. Not much research is available on how a DT infrastructure needs to be composed to realize DTs for major I4.0 use cases. Thus, this paper addresses this gap by contributing an architecture blueprint detailing DT infrastructure components and their interactions. Furthermore, an example implementation of the architecture blueprint using the Eclipse BaSyx middleware as well as an experience report and lessons learned are provided.

This paper is structured as follows: Section 2 provides an overview of the state of the art and the state of the practice. In Section 3, we provide important and motivating use cases as well as architecture drivers for these use cases. Section 4 describes our architecture blueprint and details components and their interactions.

ECBS 2021, May 26, 27, 2021, Novi Sad, Serbia
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9057-6/21/05…$15.00
https://doi.org/10.1145/3459960.3459978

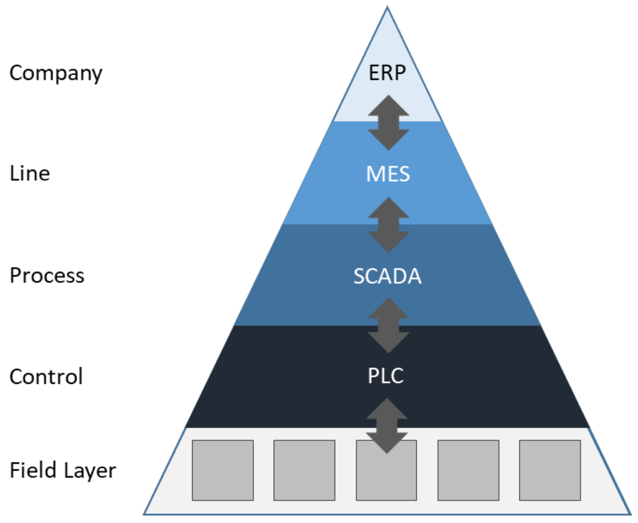

Figure 1: Architecture of the Automation Pyramid

Section 5 provides an example implementation of the architecture blueprint using Eclipse BaSyx, an I4.0 open-source middleware. Additionally, an experience report and lessons learned are presented. An overview of related work is given in Section 6. Finally, in Section 7 we draw conclusions and provide pointers for further research.

2 STATE OF THE ART & STATE OF THE PRACTICE

2.1 Industry 4.0 & Digital Twins

I4.0 introduces end-to-end communication into the manufacturing industry. Today, manufacturing devices already collect large amounts of sensor data, which is only available on the PLC layer. Additionally, data processing is only possible along the layers of the Automation Pyramid (cf. Fig. 1), which refers to the strictly separated layers and protocols in automation systems. Data use is therefore limited to one machine. In contrast, I4.0 advocates a peer-to-peer architecture that enables cross-layer interaction, which, for example, enables enterprise resource planning (ERP) systems to directly interact with sensors, e.g., to track the quality of manufacturing processes and autonomously react to loss in quality.

Cross-layer interaction requires a structured approach for representing and exchanging data. The concept of DTs, whose origin is in the testing of avionics systems [9], is a key concept for cross-layer communication in I4.0 [10]. A DT is a representation of the state of a physical entity. It enables unified access to asset data and services. As defined by the IIC [11], *"a digital twin is a formal digital representation of some asset, process or system that captures attributes and behaviors of that entity suitable for communication, storage, interpretation or processing within a certain context"*. Thus, in the context of this paper, a DT is understood as a data model describing relevant properties of the assets. Additionally, further models such as simulation models or 3D models may be included in the DT. Thereby, an infrastructure that supports distributed DTs enables the deployment of single DTs with distributed parts.

DTs should describe all relevant assets in an I4.0 production. Assets are, for example, products to be produced, work pieces, devices, and the manufacturing process. They aggregate data generated in the physical world, enable experimenting with this data in digital representations of I4.0 assets, and provide insights that may be deployed back to the physical world as updated configurations. Thus, the data provided by DTs enables plant orchestration and autonomous optimization.

2.2 Architecture of Plant Automation

Currently, automation in industrial production plants is implemented using the "automation pyramid" reference architecture as defined by IEC 62264 [4] (cf. Fig. 1). This architecture consists of separate layers with defined interaction points. Programmable Logic Controllers (PLCs) execute cyclic programs to implement specific production steps. Supervisory control and data acquisition (SCADA) realizes the distributed control of several PLCs. Manufacturing Execution Systems (MES) manage the overall production by controlling high-level manufacturing steps. Enterprise Resource Planning (ERP) systems manage resources like the number of items in stock. Cross-layer interaction in the automation pyramid is difficult, as every layer can only access information that is provided by lower layers. Available and ready-to-use I4.0 solutions therefore often connect edge devices to processes to collect data and forward this data to analysis backends. While this works for isolated applications, like the creation of dashboards and predictive maintenance, this approach only provides data to selected analysis applications. It keeps all data separate from assets, and keeps the layers of the automation pyramid intact. Process changes consequently still require modifications in all layers, and the integration of additional applications might require the inclusion of new edge devices.

An architecture supporting I4.0 solutions would enable peer-to-peer communication in all layers of the automation pyramid, thus breaking the layered architecture. This would require harmonized communication endpoints, mutual understanding of data, and the ability to bridge communication protocols. This architecture change, however, would only address the data acquisition part and would not directly improve changeability.

Programmable Logic Controllers (PLCs) are at the bottom layer of the automation pyramid. They execute cyclic programs that implement production steps. The lack of virtual communication requires engineers to change PLC programs to implement manufacturing process changes. This may lead to numerous side effects: Code changes may affect execution times and therefore violate real-time constraints of the PLC program. Predecessor and successor devices of the changed device depend on fixed cycle times and timed inter-PLC communication through networks. In consequence, a change of a single PLC may produce unforeseen side effects [12], leading to downtimes and therefore to a loss in time and money.

Sophisticated I4.0 applications, such as lot size 1 production, therefore require an architecture change that incorporates end-to-end communication for all artifacts. Currently, several reference architectures exist for I4.0. The *Industrial Internet Reference Architecture* [13] is not only a reference architecture for I4.0, but also addresses other domains like energy, smart cities, and healthcare.

It presents the architectural viewpoints of business, usage, functionality, and implementation. The *Reference Architecture Model Industry 4.0* [14] defines a three-dimensional model decomposed into hierarchy levels, lifecycle and value stream, and layers. It is a framework for creating a common understanding among stakeholders and for identifying gaps and solutions for production systems. The *Stuttgart IT Architecture for Manufacturing* [15] decomposes its architecture into three middlewares: integration, mobile, and analytics, with a focus on service-oriented architectures.

2.3 Quality-driven Specification of Architecture Drivers

The development of a new architecture has to address numerous stakeholder needs, which have to be formulated and formalized. Architecture drivers support the formalization of stakeholder needs regarding architecture decisions: Architecture drivers are a particular set of requirements classified as new, risky, or expensive to implement or maintain [16] and therefore significantly affect the architecture. The specification of architecture drivers must be precise enough to enable the architecture to properly reason about adequate architecture solutions to address them. In this regard, we adopted the approach proposed by Knodel and Naab [16], who claim that architecture drivers must be specified in terms of: (i) the environment or condition in which this driver occurs; (ii) the event that stimulates the occurrence of the driver; (iii) the expected response of the system to the driver event; and (iv) the quantifications associated with the three previous aspects. Each of these measurable effects indicates whether the driver is addressed by the architecture.

The systematic specification of architecture drivers must indicate the quality aspects that characterize the system functions described in each architecture driver specification. For instance, instead of claiming that the light barrier sensors shall transmit signals to the roller conveyor of the production plant, we might characterize the system function by the degree of security of the signal transmission (e.g., the maximum tolerable delay). In this way, the architecture drivers specify not only the what but also the how well (i.e., with what degree of quality), thereby providing proper information for the selection of an adequate architecture solution for each architecture driver. To this end, we adopted the quality characteristics described in ISO/IEC 25010 Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality model [17] to characterize the I4.0 architecture drivers described in this paper. ISO/IEC 25010 comprises the following quality characteristics: Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability. Each quality aspect is further refined into sub-aspects. For instance, Security is further refined into Confidentiality, Integrity, Non-Repudiation, Authenticity, and Accountability.

3 INDUSTRY 4.0 USE CASES

A DT structure and infrastructure for I4.0 has to support key use cases. Based on the authors' experiences and supplementing material, e.g., from the "Plattform Industrie 4.0"[1] [18], we document two recurrent use cases as representatives of the drivers that serve as a basis for deriving the architecture blueprint. Each is based on its own architecture driver specifications (cf. Table 1 and Table 2). Due to the page restrictions for this paper, these two use cases were picked to representatively illustrate the requirements for deriving this I4.0 architecture blueprint. Further I4.0 use cases are documented online[2]. The overall structure of the architecture drivers is as follows: (i) Quality Aspect describes which aspect of the ISO 25010 standard is reflected in the driver; (ii) Environment describes assumptions about the environment; (iii) Stimulus describes the trigger for the change reflected in the driver; (iv) Response describes the expected outcome; (v) Quantifications describes relevant quantifications of aspects impacting quality attributes of the overall scenario.

As a central I4.0 use case, end-to-end communication spanning all layers of the automation pyramid will be evaluated in Section 3.1. Additionally, plug'n'produce will be elaborated as another use case in Section 3.2.

3.1 End-to-End Communication

Enabling end-to-end communication is one of the main goals of I4.0 [19]. I4.0 advocates a peer-to-peer architecture that enables cross-layer interaction to allow flexible use of data as the basis for self-adapting processes. Asset Administration Shells (AAS) [20] provide common data structures and a defined type system and are therefore a valid technical foundation for DTs. AAS submodels enable the definition of tailored data structures and services to describe and represent specific production assets. All assets can access the AAS regardless of their own location within the automation pyramid. This makes the Industrial Internet of Things possible. A common technical requirement in control systems is maximum data latency, which is often constrained to a maximum value. We therefore adapted the end-to-end communication architecture driver initially proposed by Antonino et al. [6] to also reflect access within the same layer of the automation pyramid (cf. Table 1), so it also contains quantifications of latency requirements.

3.2 Plug'n'Produce

Plug'n'Play (PnP) describes the ability of a system to enable automatic discovery and configuration of a newly plugged-in component without the need for human intervention. Plug'n'Produce extends this concept to the domain of manufacturing. The common objective is to reduce the effort required to install new or additional devices into a new or existing infrastructure as well as to reduce downtimes. Koziolek et al. [21] identify nine phases for "Plug-and-Produce", classified into physical device installation in phases one to three and configuration and system integration in the remaining ones. All nine phases can be further decomposed into 51 steps in total. The first phase is only necessary when an existing device is replaced. Thus, it is not applicable when a plant is commissioned for the first time. Phases 2-8 focus on the physical integration of the device into the system while phase 9 focuses on integrating the device into the plant's IT system. In addition to the identification of the nine phases, the authors propose a Plug'n'Produce architecture called OpenPnP based on OPC-UA. OPC-UA serves as

[1] The "Plattform Industrie 4.0" is a German entity focused on defining concepts for Industry 4.0. For example, it defines the concept of the Asset Administration Shell.

[2] https://wiki.eclipse.org/BaSyx_/_Applications

TABLE 1: ARCHITECTURE DRIVER DESCRIBING INFORMATION EXCHANGE BETWEEN SYSTEMS OF THE AUTOMATION PYRAMID.

Driver	**Information exchange between systems of the automation pyramid**	**Quantifications**
Quality Aspect	➢ Compatibility (Interoperability) ➢ Performance efficiency (Time behavior)	
Environment	➢ Several devices and products exist. ➢ Applications exists on a higher layer, e.g., a dashboard. ➢ DT exists for each production entity and the production management.	➢ Number of devices ➢ Number of products
Stimulus	➢ A device has information about it being updated; e.g., a change in a critical sensor parameter like energy insufficiency or overheating.	➢ New time stamp Value != Old time stamp value
Response	➢ The changed information is propagated to the DT. ➢ Another device or application retrieves the information and, for example, sets a control point.	➢ Number of direct communications between physical devices and other applications = zero ➢ Latency requirement violations = zero

TABLE 2: ARCHITECTURE DRIVER DESCRIBING THE PLUG'N'PRODUCE USE CASE.

Driver	**Plug'n'Playability of Devices**	**Quantifications**
Quality Aspect	➢ Compatibility (Interoperability) ➢ Maintainability (Modifiability)	
Environment	➢ The production system has been deployed. ➢ A DT IT infrastructure is running. ➢ The semantics of the new device are already specified and can be interpreted by the enclosing infrastructure.	➢ Number of devices
Stimulus	➢ An additional device is installed in the described environment. ➢ The physical device is set up, including its calibration and configuration of basic communication (i.e., phases 1 to 8).	
Response	➢ The system discovers the DT infrastructure. ➢ It makes its DT available to the system, e.g., by uploading it to a server. ➢ The DTs can be found and accessed by other applications or devices.	➢ Integration time of device for phase 9 < 1 minute ➢ Manual interventions = 0

the basic communication technology from and to field devices and connected components. OpenPnP also uses the built-in mechanisms of OPC-UA for device registration and discovery.

Building on this work, we describe an architecture driver focusing on phase 9 in Table 2. The context includes preconditions and assumptions made in order to set up a common understanding of the environment. It is followed by the steps needed to trigger the response of the system.

4 ARCHITECTURE BLUEPRINT

The architecture blueprint depicted in Fig. 2 was derived to address I4.0 use cases like the ones described in Table 1 and Table 2. These abstract blueprints are meant to be utilized as reference models to instantiate specific architectures for individual contexts. For this purpose, we will also discuss instantiation for each of the introduced components.

In a nutshell, the architecture blueprint consists of (i) the *Digital Twin Registry*, which provides information about the existing DTs, e.g., of devices or products, and how to reach them; (ii) the *Digital Twin Cloud Server*, a public or private cloud that stores the DTs and provides an API offering means to upload and manipulate them; (iii) the *Edge Node*, which provides DT data that cannot be stored in the cloud due to network performance constraints like throughput or latency; and (iv) the *I4.0 Application*, which implements key features such as job scheduling and services orchestration. The detailed descriptions of each of these elements are described in Sections 4.1, 4.2, 4.3, and 4.4, respectively. The interactions among the architecture blueprint elements are described in Section 4.5.

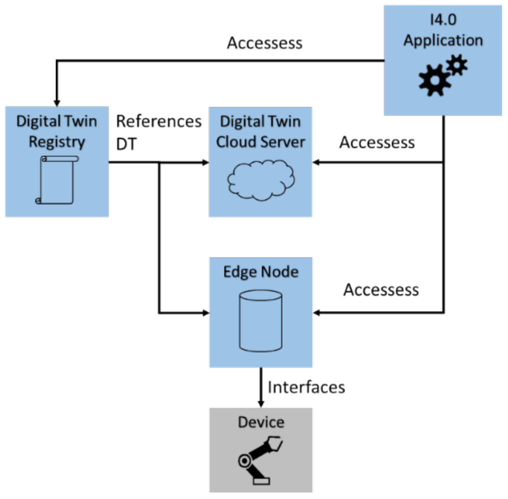

Figure 2: Architecture Blueprint

4.1 Digital Twin Registry

The *Digital Twin Registry* acts as central entry point to the DT system. It is used by I4.0 applications to (i) look up available DTs and (ii) retrieve access information, e.g., network endpoints. This concept is similar to the service registry used in classical service-oriented architecture. However, instead of describing only services provided by the different entities in the system, the *Digital Twin Registry* provides additional information about system participants, e.g., semantic descriptions of the asset type or services offered. This information enables I4.0 applications to find entities with specific properties. A service orchestrator, for example, can find all entities that provide specific manufacturing services. The *Digital Twin Registry* maps IDs to DT descriptors. IDs must be unique in a given context, so they must either be globally unique or, e.g., a combination of a product type and a serial number. The same DT may be reachable through multiple IDs. Optional additional API functions include an event mechanism to inform listeners about the connection of new assets in the system and about assets that have left the system.

The *Digital Twin Registry* can be extended by the *Directory* and the *Cascading Registry/Directory*, which will be described in the remainder of this subsection.

4.1.1 Directory. Depending on the type of an I4.0 application, a different set of properties may be relevant for the selection of a DT. Using the simple registry API described above, the application may retrieve all DTs and browse them to learn about their properties. However, this will lead to a large communication overhead. This is addressed by the Directory extension of the Digital Twin Registry. It extends the API of the registry with the ability to search for DTs with defined properties, e.g., finding all DTs of products ordered by a specific manufacturer. A simple implementation of this concept is the Tagged Directory: Assets register with this directory and provide a descriptor containing tags as descriptions of important asset properties, e.g., supported communication protocols and the asset manufacturer. Thus, an application may browse for precisely those assets that are tagged with specific properties. The definition of a standard set of tags is beyond the scope of this paper. Instead, we assume that tags appropriate to the specific context will be defined by users of the architecture.

The Tagged Directory extends the API defined by the Digital Twin Registry with methods for tag retrieval. This tag retrieval can allow arbitrarily complex tag requests, e.g., by supporting a Boolean expression of tags. For example, a tag query of "Device && OPC UA && !Mill" will return the DTs of all devices supporting OPC UA that are not mills.

4.1.2 Cascading Registry/Directory. In many cases, a single registry/discovery server might not be sufficient. A realistic I4.0 deployment will consist of a large number of devices, products, and other assets. The ability to access remote devices requires access to potentially all DT descriptors. A single registry/directory is not feasible in this case. We therefore propose a cascading structure that reflects a registry/directory as shown in Fig. 3. This structure is similar to the Internet's DNS mechanism, which maps a human-readable address to an IP address by recursively resolving the address parts.

This concept requires a structured identifier. For example, CompanyX:Plant1:Line5:MillA enables a recursive resolution by forwarding the request to the registry of CompanyX, which forwards it to the respective Plant1 and Line5 registries. Finally, the Line5 registry returns the access path to the DT of MillA. Besides the requirement on the structure of the identifier, this variant is transparent to an outside application[3].

4.2 Digital Twin Cloud Server

The *Digital Twin Cloud Server* provides the capability to upload and store DTs persistently at runtime, rather than hosting it in-memory on a device. This guarantees that a DT is available even in the event of a device failure. This enables efficiently retrieving information about the failed device and ordering its replacement, for instance. The cloud server may be part of a private or a public cloud.

To support this, an API enabling DTs to handle, i.e., to upload, delete, and manipulate DT data, has to be provided. In most cases, multiple *Digital Twin Cloud Servers* are necessary, e.g., to store product DTs close to manufacturing lines and to reduce network traffic.

4.3 Edge Node

Similar to the *Digital Twin Cloud Server*, an Edge Node is also able to host a DT. In contrast to the *Digital Twin Cloud Server*, it does not support the upload of DTs at runtime. Instead, it provides preconfigured DTs, for example with sensor data and digital nameplates. A device DT is provided for modern smart devices by the device manufacturer or by a third party. Edge nodes host DTs (or parts of a DT) that should be close to the device and the manufacturing line, e.g., because they require low-latency communication or operate on raw sensor data, which quickly overloads the network.

Edge nodes furthermore enable the retrofitting of existing devices, i.e., the integration of older devices into an I4.0 system. Currently, retrofitting typically focuses on enabling only communication with legacy systems; an example are PLC controllers that use field bus communication [22].

In a DT infrastructure, retrofitting does not only need to address the aspect of communication but also, for example, data and service models, which are imperative for DTs. This includes (i) setting up the DT and (ii) connecting it to the legacy data source.

4.4 I4.0 Application

The *I4.0 Application* leverages the DTs provided by the DT system to implement key business functionalities. A novel application is service-based production. Devices describe offered services together with quality and cost attributes in their DTs; products describe required services together with quality and cost constraints. Based on descriptions in DTs, scheduling applications generate optimized production plans by defining when and how a specific product will be produced. Another common application for DTs is predictive maintenance, where data from the past is used to build simulation models that predict critical failures based on real-time device data [23].

This illustrates the necessary coordination between application development and DT development. The application has to be tailored to the DT system to support the interaction patterns (e.g., as

[3]This proposed identifier structure is compatible with the format described in "Details of the Asset Administration Shell – Part 1" by *Plattform Industrie 4.0* (https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.html, pp. 37-40)

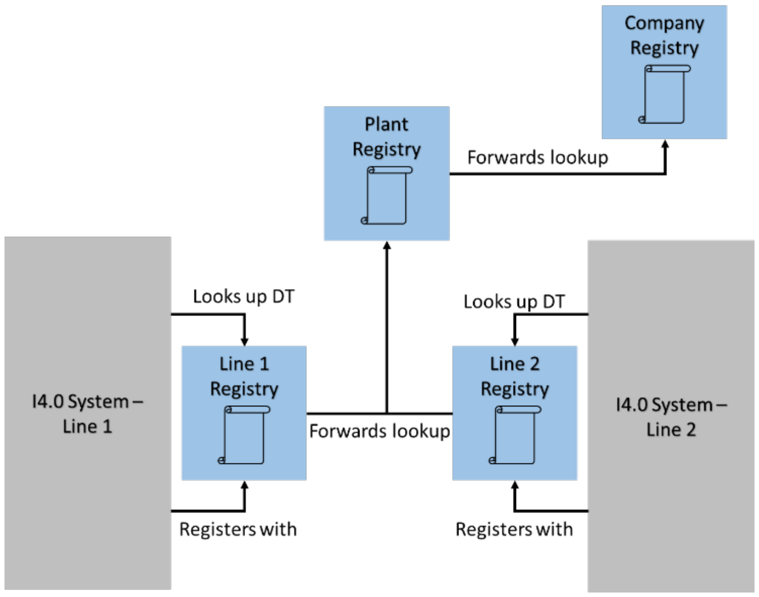

Figure 3: Cascading Directory Infrastructure

described next in Section 4.5) as well as being tailored to the chosen DT structure.

4.5 Interactions among the Architecture Blueprint Elements

The interactions among the architecture blueprint elements are based on two interaction patterns: *Digital Twin Registration* and *Digital Twin Lookup*.

4.5.1 Infrastructure Discovery. For Plug'n'Produce, it may be beneficial to provide automated discovery of cloud servers and the registry to minimize human involvement. In the case of the Internet, there already exist proven technologies for the discovery of infrastructure components. For example, multicast DNS[4] (mDNS) utilizes multicast frames to address every entity in the local network for discovery. This can then be leveraged to perform the bootstrapping process of discovering the DT registry/directory and the optional cloud server in a local network.

4.5.2 Digital Twin Registration. Each time a new asset is created, its DT has to be registered with the *Digital Twin Registry* to make its descriptor available to applications. The following steps are therefore mandatory for asset registration:

1. **Infrastructure Discovery.** In this step, the infrastructure is discovered. Alternatively, this information can be preconfigured.
2. **Digital Twin Upload.** If applicable, the DT is uploaded to the discovered *Digital Twin Cloud server*. To do so, the provided server API is used.
3. **Digital Twin Registration**. Either the DT or parts of it are registered at the registry with its descriptor. As described in Section 4.1, this descriptor contains the access information and possibly additional data (e.g., tags).

4.5.3 Digital Twin Lookup. By performing the registration process, it is then possible to also locate assets from remote locations if cascading directories are used. Similarly, the lookup interaction can be performed by an application accessing the system in the following way:

1. **Registry Discovery.** In this step, the *Digital Twin Registry* is discovered. Alternatively, this information can be preconfigured.
2. **Descriptor Retrieval.** The DT descriptor is retrieved from the *Digital Twin Registry.* As described in Section 4.1.2), this retrieval may involve multiple registries that forward the requests to other registries in the network.
3. **Digital Twin Access.** There are two options for accessing the DT: either the DT has been uploaded to the *Digital Twin Cloud Server*, or it resides within an *Edge Node.* Although these options influence the location of the asset's DT, the application accessing it does not have to differentiate. Since it only accesses an endpoint, it can directly access the DT as long as its interface is made available to the application.

5 ARCHITECTURE BLUEPRINT IMPLEMENTATION & EXPERIENCE REPORT

In order to provide a tangible model of the discussed components and demonstrate a possible way for realizing the concepts within a given scenario, we implemented the described DT infrastructure in the context of the Eclipse BaSyx[5] middleware. It provides concepts needed to implement I4.0 in production systems. A major

[4]https://www.rfc-editor.org/info/rfc6762

[5]https://www.eclipse.org/basyx

artifact of Eclipse BaSyx is the DT, which is realized with the AAS [20]. Ongoing standardization activities aim at creating a common meta-model and runtime interface for the AAS. In particular, the Eclipse BaSyx middleware was used for implementing AAS and their submodels and for creating the infrastructure components.

5.1 Demonstration Scenario

In the demonstration scenario, we assume that an oven has to be commissioned into the plant. The plant IT system supports AAS and provides the necessary infrastructure according to the blueprint specified in Section 4. This infrastructure includes an *AAS Tagged Directory* and a dedicated *AAS Cloud Server* enabling the upload of AAS and submodels (cf. Fig. 4).

It is assumed that the oven provides its own DT as an AAS as well as accompanying submodels. In detail, the device provides the following entities:

1. **Sensor AAS:** The AAS of the sensor, containing meta-data and referencing the associated submodels. It is hosted on the *AAS Cloud Server*.
2. **Sensor data submodel:** Provides current sensor data, i.e., is refreshed with every update of the sensors themselves and is hosted on the *Edge Node*.
3. **Documentation submodel:** Contains the documentation data of the oven. For example, a maximum temperature is given. It is hosted on the *AAS Cloud Server*.

The device performs the following steps:

1. **AAS/Submodel Upload:** Upload the AAS and its documentation submodel onto the AAS cloud server.
2. **AAS/Submodel Registration:** Register the AAS and both submodels with the tagged directory using the *SensorDevice* tag.

Then a plant monitoring application performs the following steps:

1. **Descriptor Retrieval:** Retrieve the descriptor of all AAS tagged with *SensorDevice* and access the descriptor of the newly registered AAS.
2. **AAS/Submodel Access:** Retrieve information about the oven by accessing the newly added AAS and its submodels, such as the maximum temperature documented in the documentation submodel.

5.2 Instantiation Strategy

Several gaps have to be addressed when instantiating our architecture blueprint to a real-world application. In the following, we will detail these gaps and describe how we filled them for the demonstration scenario.

5.2.1 Communication System. Our blueprint does not specify any communication protocol or system. Any communication system used should support end-to-end communication. Master/slave protocols or any other communication protocols that do not support every node initiating communication reduce the benefits of the architecture blueprint. Additionally, if discovery of infrastructure components is intended, a discovery mechanism (e.g., multicast) has to be supported. In the authors' experience, Ethernet-based protocols, e.g., HTTP/REST or OPC UA, support the architecture best.

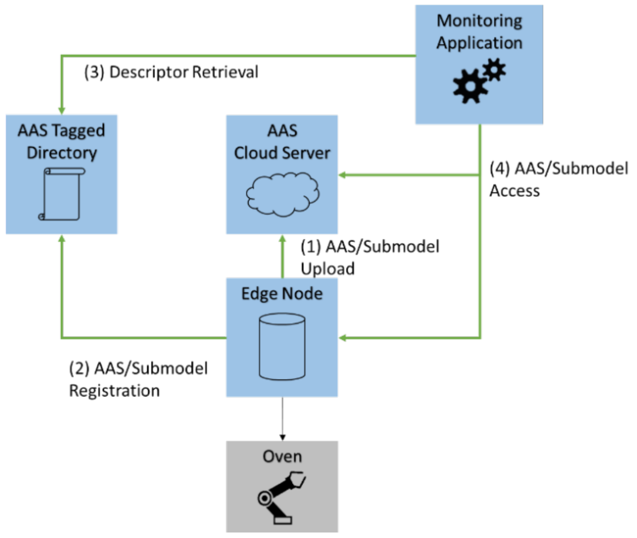

Figure 4: Architecture Blueprint Instantiation for the Demonstration Scenario using AAS

In the latter case, discovery and a registry/directory are already provided by the communication infrastructure. For our prototypical implementation, HTTP/REST was chosen.

5.2.2 Digital Twins. The architecture supports arbitrary data models and thus any DT. However, in practice, I4.0 use cases greatly benefit from using data models that are commonly agreed upon or even standardized. When using standardized models, little to no tailoring of applications is necessary. Instead, an application will be interoperable with the DT's structure and semantics by design. The AAS [20] is such an example of a standardized DT and was therefore used for the prototypical implementation.

5.2.3 Descriptors. As a central point of entry, the registry can be tailored to specific use cases. As previously described, in a simple case it may solely provide a mapping from an Id to a descriptor containing an access path. However, depending on the application, additional information may be beneficial. For example, a descriptor may, in addition to the access path, provide further information about the described asset, e.g., the manufacturing services it provides or its type. The descriptor concept is easily extensible, e.g., by utilizing tagging methods as described in Section 4.1.1. In several of the authors' projects, it turned out to be beneficial to store data often requested but seldom or never changed in the descriptors to reduce requests. For example, if a DT references other (sub) DTs, it may be beneficial to reference the DTs directly from the descriptor. Thus, for the prototypical implementation, a descriptor format containing information about the AAS and its submodels was chosen.

5.3 Example Implementation

We implemented the demonstration scenario using the Java SDK of the open-source middleware Eclipse BaSyx. With it, we specified the AAS of the device and its submodels. Additionally, we used BaSyx to implement the DT infrastructure components. These infrastructure components are available as Docker Images to enable quick setup of DT infrastructures.[6] These components were used to host the AAS and the submodels. In the following snippets, we assume that the infrastructure was already discovered.

The code shown in (1) first creates an AAS and accompanying submodels for the oven. Next, (2) showcases pushing the AAS to the *AAS Cloud Server* and registering it with the *SensorDevice* tag at the *AAS Tagged Directory*. Additionally, it shows pushing the Documentation submodel to the *AAS Cloud Server* and registering both the Documentation submodel and the Sensor submodel with the *AAS Tagged Directory*. Thus, deployment of distinct parts of DTs can be split between individual submodels and the AAS.

```
/** (1) **/
// Create the oven asset
Asset ovenAsset = new Asset("OvenAsset", new
CustomId("icsa21.OvenAsset"),
AssetKind.INSTANCE);
// Create the AAS representing the oven
AssetAdministrationShell ovenAAS = new
AssetAdministrationShell("ovenAAS",
new CustomId("icsa21.OvenAAS"), ovenAsset);
// Sensor submodel creation excluded
Submodel sensorSm = . . .
// Create the documentation submodel
Submodel docuSm = new SubModel("docuSM", new
CustomId("icsa21.docuSM "));
// Create the maximum temperature property and
// include it in the submodel
Property maxTemp = new Property("maxTemp", 800);
docuSm.addSubmodelElement(maxTemp);
// Create the directory proxy
IAASTaggedDirectory directory = new
TaggedDirectoryProxy(directoryPath);
// Create a ConnectedAASManager with the registry created above
IAssetAdministrationShellManager aasManager = new
ConnectedAssetAdministrationShellManager(
directory);
/** (2) **/
// Push the AAS to the cloud server
aasManager.createAAS(ovenAAS, cloudPath);
// Register the newly added device with the "SensorDevice" tag
TaggedAASDescriptor desc = new
TaggedAASDescriptor(ovenAAS, aggregatorPath);
desc.addTag("SensorDevice");
directory.register(desc);
// Push the docuSubmodel to the cloud. The manager
automatically registers it
aasManager.createSubmodel(ovenAAS.getIdentifier(),
docuSubmodel);
// Register the sensor submodel
SubmodelDescriptor smDesc =  new
SubmodelDescriptor(sensorSM, edgePath);
directory.register(ovenAAS.getIdentifier(), smDesc);
```

[6]https://wiki.eclipse.org/BaSyx_/_Documentation_/_Components

Next, the application code is shown. The monitoring application retrieves the descriptors of all AAS tagged with *SensorDevice* by querying the *AAS Tagged Directory* (cf. (3)).

Finally, it retrieves the AAS and its submodels. For further processing, it accesses the *maxTemp* property and reads its value (cf. (4)).

```
// Create the directory proxy
IAASTaggedDirectory directory = new
TaggedDirectoryProxy(directoryPath);
// Create a ConnectedAASManager with the registry created above
IAssetAdministrationShellManager aasManager = new
ConnectedAssetAdministrationShellManager(directory);
/** (3) **/
// Get all descriptors with the “SensorDevice” tag
Set<TaggedAASDescriptor> descriptors =
directory.lookupTag("SensorDevice");
// Excluded: Filtering for new devices
// ...
// Exemplary processing of the newly added oven descriptor
IAssetAdministrationShell oven =
aasManager.retrieveAAS(ovenDescriptor.getIdentifier());
// Retrieve the documentation submodel
ISubmodel docuSM = oven.getSubmodel(new
CustomId("icsa21.docuSM"));
/** (4) **/
// Retrieve the maxTemp Property
int maxTemp = (int) docuSM.
getSubmodelElement("maxTemp").getValue();
```

5.4 Experience Report & Lessons Learned

We instantiated our DT infrastructure in various demonstrators and customer projects. It was developed in the context of the BaSys 4.0 project. There, the proposed approach is utilized by the consortia members including *Robert Bosch GmbH*, *Lenze Gruppe* and *SMS Group*. The proposed DT infrastructure and DT structure have proven to be appropriate for addressing the different use cases of these applications. We observed an improvement in the time needed for implementing I4.0 systems. By following the architecture blueprint, reusable components were implemented that could be used in various projects and demonstrators. Additionally, in our experience, the maintainability of the implemented systems improved and new devices could be integrated quickly with already existing applications. Furthermore, changing DT deployment by moving submodels from *Edge Nodes* to the *Digital Twin Cloud* or vice-versa in later stages of development did not require any change in the *I4.0 Applications*. The interaction patterns described in Section 4.5 ensured that only the entry in the *Digital Twin Registry* had to be updated. However, some concepts need to be evaluated in more detail to leverage their potential. For example, mDNS as a discovery mechanism only works under certain conditions, e.g., the device and the component to be discovered need to share the same subnet. If this is not the case, a more sophisticated discovery mechanism are needed.

While the proposed tagging system for DT registration proved to be very useful, it has its limitations. It is only able to express Boolean logic; i.e., a tag is either there or it is not. However, complex use cases may require finer-grained query support. For example, an application may want to retrieve all DTs of products manufactured between one and two o'clock of a specific day due to a manufacturing error during this timeframe. Assuming the manufacturing date is documented in the DTs of the product, a search interface can make it possible to filter them for the required attributes. However, enabling such a search interface requires preprocessing (e.g., indexing) of the DTs. Additionally, the question arises what properties should be indexed. It is expected that in practice, DTs may contain thousands of properties (e.g., the process documentation of a product). Thus, indexing of complete DTs may impose new challenges.

Last, application communication with the infrastructure and the DTs was performed based on HTTP/REST calls, i.e., explicit pull of information. However, more efficient communication would be enabled by providing an event interface. Such an interface could support events on various levels:

- Adding or removing a DT to/from the *Digital Twin Registry*
- Changing the structure of the DT (e.g., adding a new attribute)
- Changes of the DT's attributes (e.g., update of a sensor value)

6 RELATED WORK

In [24], Redelinghuys, Basson and Kruger present a six-layer DT architecture supporting a manufacturing cell and extend the idea of a DT by aggregated DTs to form a potential hierarchy. However, they do not present a technology-independent architecture. The layers comprise (i) the physical twins, (ii) controllers like PLCs, (iii) the local data repositories layer focused on OPC UA and local databases, (iv) the IoT gateways, (v) cloud-based information repositories providing current and historical data of the DT, and (vi) emulation and simulation applications. The effect of latency is identified as a limiting factor for cloud-based information repositories.

Malakuti and Grüner [25] detail various architectural aspects of DTs in IIoT systems. They focus on (i) internal structure and content, (ii) APIs and usage, (iii) integration, (iv) runtime environment, and (v) realization. For each aspect, the authors describe various mechanisms that have to be provided to address it.

An architecture of an intelligent DT is proposed by Talkhestani et al. [26]. It covers Id aspects, DT version management and their relations, as well as organizational and technical data. Additionally, a co-simulation interface is described. The authors identify several challenges of using DTs in manufacturing, such as (i) integration of interdisciplinary models, (ii) synchronization of reality and virtuality, (iii) co-simulation capabilities, and (iv) active data acquisition. They propose distinct architectures addressing these challenges but do not focus on the distribution of DTs.

Alam and Saddik [27] describe a DT architecture reference model for cloud-based cyber-physical systems. They analytically describe the key properties of their proposed architecture reference models based on the key aspects of computation, control, and communication, addressing operation modes depending on system context. Furthermore, they provide a prototypical implementation.

Leng et al. [28] describe a framework for DT-driven cyber-physical manufacturing systems. They describe various key enabling technologies like (i) data-driven cyber-physical fusion, (ii) triple-view cyber-physical synchronization and integration, (iii) decentralized self-organizing, and (iv) online parallel controlling. Furthermore, they describe remaining challenges including building a set of interoperable DT models.

Lee, Bagheri and Kao [29] propose five layers for CPS composed of (i) smart connection, (ii) data-to-information conversion, (iii) cyber, (iv) cognition, and (v) configuration. Additionally, they detail multiple advantages of implementing cyber-physical systems in factories including near-zero downtime production and optimized production planning.

Wagner et al. [20] describe the concept of the AAS and its role in the lifecycle of a plant but do not introduce an architecture. They outline the lifecycle of a legacy plant in contrast to an I4.0 plant. Based on this outline and comparison, they provide various recommendations for actions targeted at different I4.0 personas.

7 CONCLUSION & FUTURE WORK

DTs enable central I4.0 use cases. However, applying DTs requires an infrastructure that supports their use. In this paper, we have described an architecture blueprint for distributed DTs motivated by architecture drivers. In this architecture blueprint, we propose four key elements: (i) a *Digital Twin Registry* as the central interaction point for adding or retrieving DT access information; (ii) a *Digital Twin Cloud Server* enabling upload and manipulation of DTs; (iii) the Edge Node hosting DT data that cannot be stored in the cloud due to network performance constraints; and (iv) the I4.0 Application that realizes business functionalities and accesses the DT system.

We specified core interaction patterns of the architecture blueprint and provided an example open-source implementation of the architecture blueprint using Eclipse BaSyx and AAS. Finally, we presented an experience report and lessons learned while applying our architecture blueprint.

While the DT architecture blueprint outlines components and their interactions, for specific use cases, there is a need for agreed data models and descriptions to be contained in the DT. Thus, we assume that future research will focus on the specification of data models and their semantic specifications to be included in the DTs. In addition, a systematic discussion of additional architecture drivers as well as use cases and detailed instructions on how to apply the patterns in each case could further validate the results of this paper. Also, such a systematic discussion could support future technical implementations and yield further refinements. Additionally, we consider instantiation guidance on how to apply the described architecture blueprint to different contexts and use cases an interesting topic for future research. Finally, various architectural solutions based on a wide variety of I4.0 use cases have been developed in recent years and will continue to be developed. Considering this is an emerging topic, the adaptation of the concepts introduced in this paper in further real applications could be evaluated in future work, for example through case studies.

ACKNOWLEDGMENTS

This work is funded by the German Federal Ministry of Education and Research (BMBF) under grantno. 01IS19022R through the project BaSys 4.2.

REFERENCES

[1] Lasi, H., Fettke, P., Kemper, H. G., Feld, T., & Hoffmann, M. 2014. Industry 4.0. Business & information systems engineering, 6(4), 239-242.

[2] Drath, Rainer, and Alexander Horch. 2014. "Industrie 4.0: Hit or hype?[industry forum]." IEEE industrial electronics magazine 8.2 56-58.

[3] Kiel, Daniel, *et al.* 2017. "Sustainable industrial value creation: Benefits and challenges of industry 4.0." International Journal of Innovation Management 21.08 (2017): 1740015.

[4] International Electrotechnical Commission. 2003. "IEC 62264-1 Enterprise-control system integration–Part 1: Models and terminology." IEC, Genf (2003).

[5] Daniel Großmann, *et al.* 2015. "FDI-Field Device Integration." VDE VERLAG GmbH.

[6] Pablo Oliveira Antonino, *et al.* 2019. "Blueprints for architecture drivers and architecture solutions for Industry 4.0 shopfloor applications." Proceedings of the 13th European Conference on Software Architecture-Volume 2.

[7] Zheng, Yu, Sen Yang, and Huanchong Cheng. 2019. "An application framework of digital twin and its case study." Journal of Ambient Intelligence and Humanized Computing 10.3 (2019): 1141-1153.

[8] Frank Schnicke, Thomas Kuhn, and Pablo Oliveira Antonino. 2020. "Enabling Industry 4.0 Service-Oriented Architecture Through Digital Twins." European Conference on Software Architecture. Springer, Cham.

[9] Edward Glaessgen and David Stargel. 2012. "The digital twin paradigm for future NASA and US Air Force vehicles." 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA.

[10] R. Rosen, G. Von Wichert, G. Lo and K. D. Bettenhausen. 2015. About the importance of autonomy and digital twins for the future of manufacturing. IFAC-PapersOnLine, 48(3), 567-572.

[11] Definition of Digital Twin, https://www.iiconsortium.org/pdf/IIC_Digital_Twins_Industrial_Apps_White_Paper_2020-02-18.pdf, last accessed 2020/12/18

[12] V. Vyatkin. 2013. Software engineering in industrial automation: State-of-the-art review. IEEE Transactions on Industrial Informatics, 9(3), 1234-1249.

[13] Industrial Internet Consortium. 2015. "Industrial internet reference architecture." Technical Article. Available online: http://www. iiconsortium. org/IIRA. htm (accessed on 05 Mai 2020).

[14] DIN specification "91345: 2016-04 (2016) Reference Architecture Model Industrie 4.0 (RAMI4. 0)."

[15] L. Kassner, C. Gröger, J. Königsberger, E. Hoos, C. Kiefer, C. Weber, ... & B. Mitschang. 2016. The Stuttgart IT architecture for manufacturing. In International Conference on Enterprise Information Systems (pp. 53-80). Springer, Cham.

[16] J. Knodel and M. Naab. 2016. Pragmatic Evaluation of Software Architectures, Springer Publishing Company.

[17] ISO/IEC, "Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models," ISO, 2011.

[18] https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/fortschreibung-anwendungsszenarien.html, last accessed 2020/12/20

[19] Keliang Zhou, Taigang Liu and Lifeng Zhou. 2015. "Industry 4.0: Towards future industrial opportunities and challenges." 2015. 12th International conference on fuzzy systems and knowledge discovery (FSKD). IEEE.

[20] Constantin Wagner, *et al.* 2017. "The role of the Industry 4.0 asset administration shell and the digital twin during the life cycle of a plant." 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE.

[21] Heiko Koziolek, *et al.* 2019. "OpenPnP: a plug-and-produce architecture for the industrial internet of things." 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP). IEEE.

[22] Theo Lins, *et al.* 2018. "Industry 4.0 Retrofitting." 2018. VIII Brazilian Symposium on Computing Systems Engineering (SBESC). IEEE.

[23] Panagiotis Aivaliotis, Konstantinos Georgoulias, and George Chryssolouris. 2019. "The use of Digital Twin for predictive maintenance in manufacturing." International Journal of Computer Integrated Manufacturing 32.11 (2019): 1067-1080.

[24] Anro J. H. Redelinghuys, Anton H. Basson, and Karel Kruger. 2018. "A six-layer digital twin architecture for a manufacturing cell." International Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing. Springer, Cham.

[25] Somayeh Malakuti, and Sten Grüner. 2018. "Architectural aspects of digital twins in IIoT systems." Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings.

[26] Behrang Ashtari Talkhestani, *et al.* 2019. "An architecture of an intelligent digital twin in a cyber-physical production system." at-Automatisierungstechnik 67.9 (2019): 762-782

[27] Kazi Masudul Alam, and Abdulmotaleb El Saddik. 2017. "C2PS: A digital twin architecture reference model for the cloud-based cyber-physical systems." IEEE access 5 (2017): 2050-2062.

[28] Jiewu Leng, *et al.* 2019. "Digital twin-driven manufacturing cyber-physical system for parallel controlling of smart workshop." Journal of ambient intelligence and humanized computing 10.3 (2019): 1155-1166. system." at-Automatisierungstechnik 67.9 (2019): 762-782

[29] Jay Lee, Behrad Bagheri, and Hung-An Kao. "A cyber-physical systems architecture for industry 4.0-based manufacturing systems." Manufacturing letters 3 (2015): 18-23.