On the development of a component model for the realization of Industry 4.0

Luís Neto*‡¶, Gil Gonçalves*‡¶, Pedro Torres*†§‖, Rogério Dionísio†§‖

*SYSTEC, Research Center for Systems & Technologies

†DiSAC – R&D Unit, Avenida do Empresário, 6000-767 Castelo Branco, Portugal

‡FEUP, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal

§Escola Superior de Tecnologia, Instituto Politécnico de Castelo Branco

¶Email: {lcneto,gil}@fe.up.pt

‖Email: {pedrotorres,rdionisio}@ipcb.pt

***Abstract*—The fourth industrial revolution promotes Industrial Cyber Physical Systems (ICPS) as the key to achieve smart, efficient, flexible and self-organizing production plants. In a shop floor there are heterogeneous physical and logical assets that form the ICPS. But without proper communication and composition techniques the integration of these assets in ICPS is compromised. Component Based Software Engineering (CBSE) is a discipline of growing relevance for ICPS because integration and composition issues have been extensively researched in the software domain. Under the Reference Architecture for Industry 4.0 (RAMI 4.0), the Industry 4.0 Component Model inherits aspects of CBSE to specify how several industrial plant assets can form an ICPS. The technological aspects for physical assets digitalization and integration have been explored, but the I4.0 Component model lacks proposals and use cases for dealing with industrial software components. In this work we discuss the development of the Smart Component Model as a proposal for integration of software components in ICPS. Furthermore, we focus on how prediction and monitoring applications could be converted in I4.0 Components and integrated in ICPS. To sustain our proposals, we describe a real industrial case study where these developments are being applied.**

***Index Terms*—Component Based Software Engineering; Component I4.0; Smart Component; Cyber Physical Systems; Predictive Maintenance; Condition Monitoring**

I. Introduction

The assets in a traditional production line are strongly coupled in sub-systems to accomplish specific physical processes. The combination of these sub-processes results in production of one or several final products. Each asset can be characterized by a different level of abstraction; a lower level asset could be a proportional-integral-derivative (PID) peace of software or a sensor feeding the PID instance, while a higher level asset could be an entire manufacturing execution system (MES) or a whole production cell. Components can also be characterized, among other properties, by their granularity; a lower level component like the sensor can be considered as atomic, while a higher level component like the production cell can be considered as a composite component due to high amount of sub-systems that compose it. One of the Industry 4.0 objectives is to transport this view of the factory and apply it to realize the vision of reconfigurable manufacturing systems. Components with various degrees of abstraction and granularity could then be quickly composed to respond to new necessities, among other positive outcomes [1].

This discrete view of systems has been extensively explored in the software domain for decades, the reader should refer to [2], where several aspects of component-based software systems are allready discussed. One objective of CBSE is to create applications with heterogeneous components in a plug&play fashion [3], therefore, standardized abstractions and architectures are needed to support this paradigm. In industry the same composition problem is known as plug&produce [4], [5]. This concept deals with integrating production line assets in a automatic way and has been explored in several research works [6]. The Component I4.0 model [7], [8] is essentially a standardized proposal to address plug&produce in the context of Industry 4.0. This proposal builds on top of several consolidated standards to create a component-based ICPS transversal to physical and digital assets.

A component model is a set of rules that dictate how components can be built and composed to form a system [9]. These rules include, at least the definition of syntax and semantics that allows the components be interfaced. An ICPS can be considered a component model. The components of a CPS need to be integrated and able to understand each other, otherwise the ICPS could not be established. As in a component model, the elements of an ICPS must follow certain communication and integration rules to be able to cooperate. Several works [10], [11] explore this perspective to suggest a component models for CPS. Other component model approaches regarding embedded systems and industrial applications are surveyed in [12]. In this work we describe the first steps towards a component model whose objective is to constitute a solution for digital assets in industrial environments.

The rest of the paper is organized as follows: Section II introduces the Component I4.0, a Component Model applied to Industry 4.0 components. In Section III, addresses how the Smart Component Model, joins the Component Based Software Engineeing theory and the Component I4.0 requirements to realize a component model for Industry 4.0. Section IV, describes the case study and details the problems of a company with old machines. We also address how each machine, sensor

and actuator considered is mapped in a component abstracted by the Smart Component. In Section V, we discuss future work plans to process data from these machines and how to make it useful for other components in the company ICPS.

II. Component I4.0

One example of how a component model can be applied to several disciplines is the concept of Component I4.0, proposed by German Electrical and Electronic Manufacturers Association's (ZVEI's) and Plattform Industrie 4.0 [7]. In this vision, an Asset Administration Shell (AAS) abstracts one or several assets, by being able to map the asset properties to different information models. An asset abstracted by an AAS becomes a I4.0 Component, because all its properties and functions are mapped into standardized information models and the interfaces between i4.0 Components are standardized. Therefore, all AAS share standardized communication stacks and semantics, which makes it possible to integrate all I4.0 Components in a ICPS.

If we consider a 3D Printer as an example of a physical asset, we can describe it according different properties. Each property can be mapped in specific technical domain, e.g. mechanical, electrical and software domains. In [13], there are several examples of how properties, data and functions from several machines can be mapped in standardized information models, according with its technical domains. In [8], Ye and Hong present and describe in detail the steps to convert assets of lab scale manufacturing system in Components I4.0; and how to integrate them in a CPS. This component model traverses several technical domains to create an ICPS composed of I4.0 Components.

III. Smart Component Model

The Component I4.0 concept established itself as a reference component model for the realization of Industry 4.0, under the scope of RAMI 4.0. While ZVEI's and Platform Insdutrie 4.0 published several technical specification documents [7], [13]–[15], and the academia is also addressing this research topic [16], [17]. The Smart Component model that we propose addresses some aspects related with the software domain that we consider are still missing.

ZVEI released in October 2018 a position paper involving an use case for I4.0 software components [18]. This position paper states some of the common issues of software components models, such as standardization of interfaces and dependency problems. The document also barely details the use case and does not describe technical proposals. In section IV, we present a case study of a real industrial scenario that highlights some aspects related with software components in ICPS. We think the problems described in the case study, and the Smart Component model proposal to those problems, might constitute a more concrete contribute for software integration in the Component I4.0.

- The shop floor is composed of software and machines from different vendors and generations, therefore is not straightforward to acquire data from these assets, to integrate software components to process this data, and to convert these software components in I4.0 Components.
- There exists a growing trend, that we also tried to address in this case study, to introduce machine learning techniques for condition monitoring [19], [20], process parameter optimization [21]. These techniques are developed in high level programming languages, such as python. This contrasts with the typically low level languages used to program for field devices and industrial communication protocols. To integrate these techniques with low level systems requires a considerable effort.
- When dealing with industrial machine controllers, software is typically closed source and provides scarce interfaces, information is hard to extract and integrate with other systems.

The Smart Component is a on-development component model and framework that addresses these problems and tries to constitute an AS for inclusion of software in Industry 4.0 based CPS. The problem with language independence is that a lot of scientific and open source machine learning code is produced in languages that are not exactly tailored for embedded systems. In this work we address mainly how the Smart Component Model (SCM) provides a way to integrate such OTS (of the shelf) software components.

In Fig. 1 a functional block diagram of the Smart Component is illustrated. The right side of the figure shows the *Component Repository*, it contains all components that can be instantiated by the *Component Infrastructure*. This repository is local and can be updated with new components. The component runtime, was running on top of a Java Virtual Machine (JVM) in previous versions of the Smart Component [22], [23]. The JVM approach has a portability advantage, but it lacked performance and it proved very difficult to integrate components built in other programming languages. To tackle these issues, the Linux Kernel is now used as runtime environment. This approach allows to integrate components built in different programming languages, due to the Kernel inter-process communication (IPC) mechanisms based on files, pipes and shared memory pages. The Smart Component block that allows to manage the running components is built as a Linux Kernel module. This allows to extend any Linux based distribution with the proposed Smart Component Model capabilities.

The Smart Component interfaces, illustrated in Fig. 2, specifies the access points to a component. These interfaces data types are standardized to enable reuse and allow components to inter-operate locally and remotely. The data types used by the interfaced are *OPC UA Data Types*, specified in the *OPC UA Base Model* [24]. It is important to note that an interface offers no implementation of any of its operations. Instead, it merely names a collection of operations and provides the descriptions and protocols for these. Components can import and export interfaces; imports declare what a component needs form the environment, exports declare what a component offers to the environment.

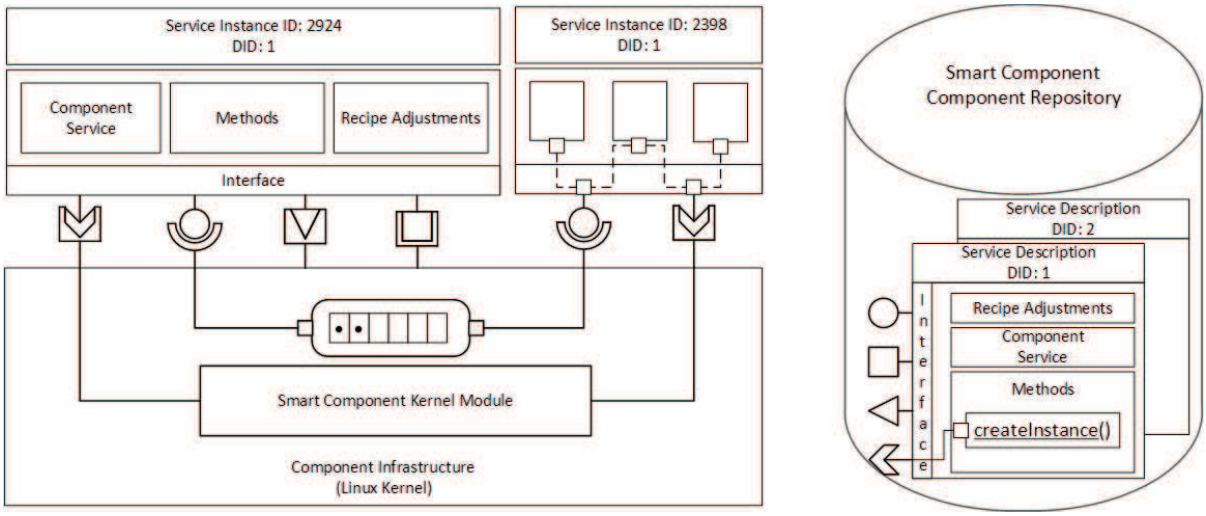

Fig. 1. Functional block diagram of *SmartComponent* (Adapted from [2])

A. Smart Component Kernel Module

As previously discussed, the decision change the runtime environment of the Smart Component Framework (SCF) from a JVM to a Linux Kernel brings some advantages. In this subsection we discuss how the Smart Component Kernel Module (SCKM) complies with the Smart Object Component Model (SOCM), how it keeps track of executing components and how component interfaces are managed. The SCKM maintains a Smart Object Self Description (SOSD) structure that describes all components and their relations. *SOSD:Device* objects are data structures which hold asset properties. *SOSD:ServiceInstance* objects, are instances of components, for the Linux Kernel, they are regular processes. The data structure that holds SOSD related info also contains pointers to the kernel process descriptor structure. This way, the SCKM keeps track of the running components, and because it resides in the kernel space, it can access the component memory regions. This is fundamental to establish the relationships between components that are specified by the SOSD structure. This characteristic also makes it possible to develop a component compliant with the SOCM specification with minimal intervention from the developers. Each compliant component must have a correspondent *SOSD:ServiceDescription* object, where all interfaces and properties (see Fig. 2 must be declared. A set of functions will be available for the developers to register the program variables within the SCKM. Each of the declared component interfaces must be mapped into a program variable by the developer. This way, with minimal effort, the programmer can read/write data directly from/to regular program variables, as if the other components were part of the same package and written in the same programming language. The restriction imposed so far is that the component interfaces must only adhere to the *OPC UA Data Types*, specified in the *OPC UA Base Model* [24]. Sometimes it happens that a closed piece of software (e.g. a device driver or database connector) must be integrated into the design. One way to achieve this, using the SCF, would be to create a simple interface component, which would connect to closed software using the available interfaces, and then, to map those channels into variables described in a *SOSD:ServiceDescription*.

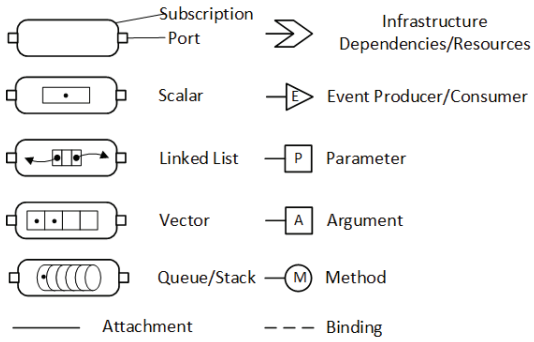

Fig. 2. Component interfaces.

Each interface between components is handled by a *SOSD:Subscription* structure, see Fig. 2. This structure corresponds to a shared memory region that is attached to the participating processes address space. As illustrated in Fig. 2, the subscription structure can hold: a scalar, a linked list, a vector or a queue/stack. The *Port* blocks are illustrative, these encode information such as the read/write permissions and synchronization mechanisms. The *SOSD:Subscription* class properties are used to encode all information related with *Ports*.

B. SOSD

The Smart Object Self Description information model was created to describe the software components available to instantiate, or running, in a certain Smart Component; to relate

these components with other assets; and also design applications based on the relation of components themselves [25]. It works like an Architecture Description Language, because the *SOSD:Device* class purpose is to give context and hold properties of the assets. In terms of the CPS, this allows to identify what software components are associated to other assets. In terms of software components, these can read the associated *SOSD:Device* structure to get/set specific parameters useful for its service. As an example, the *Inductive* object in Fig. 3 - which corresponds to the inductive sensor in the case study architecture (Fig 4) - includes properties such as the *digital pin number*, that keep the address associated with the pin to which the sensor is attached. This property can be retrieved by the *Digital Signal Acquisition* component, so it knows what address must be read to retrieve the sensor reading. A *SOSD:ServiceInstance* class, correspond to an instance of a *SOSD:ServiceDescription* class, running in a certain SmartBox. In Fig. 3, are illustrated two instances of the *Digital Signal Acquisition* component, these are sampling the Inductive Sensor and the Light Tower signals, which map the respective assets in physical architecture of the case study, Fig. IV. A component instance is identified by an unique ID, and the details of that component can be obtained through a DID (Description ID), which establishes the correspondence between *SOSD:ServiceInstance* and *SOSD:ServiceDescription* classes. The *ServiceDescription* class describes the software components and respective interfaces. This class is also useful to manage components in the repository (Fig. 1), the properties declared can be used to compare versions and query for certain types of components.

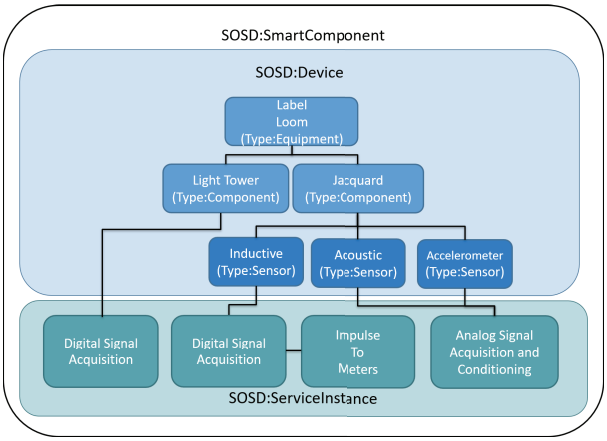

Fig. 3. Map of the case study in SOSD classes.

C. OPC UA

So far we discussed the SCM and some technical aspects of how the SCKM manages components. In this subsection we address how a remote SmartBox - running a component design on top a Linux Kernel with the SCKM - is managed. As the Component I4.0 technical documents specify, OPC UA is the standard architecture for communication between I4.0 Components. This was the major reason to establish a tight integration between this architecture and the SCF, but there are other advantages, as described in [6]. The OPC UA address space is based in a very rich information model and applies an object oriented approach to how information is structured, encoded and related. An information model can be added to an OPC UA server if its objects are built on top of OPC UA models. This allows for any OPC UA Client to be able understand and de-serialize information of a node in OPC UA Server without prior knowledge. The SOSD model, as previously stated, was built on top of the *OPC UA Base Model*, so it can be mapped in a OPC UA Server address space. Because of this integration, the SOSD structure held by the SOKM is tightly integrated with an OPC UA Server. Changes made in the design by an OPC UA Client are reflected in a remote Smart Component Runtime, which is a straightforward way to create new applications or to reconfigure the design in execution. Another possible advantage of this tight coupling, is that a certain design can request or subscribe information of another OPC UA Server. If that server is also integrated with a SCKM, this would constitute a transparent way of building distributed applications, taking full advantage of a CPS based in I4.0 Components.

IV. Case Study

This case study was explored in the context of the Portuguese research project PRODUTECH-SIF [26]. The validation of the SmartComponent is being performed on the shop floor of a textile unit that produces clothing labels. This factory in its industrial plant has machines with different levels of technology, old without any type of sensing and modern with some sensors, PLCs for control, local data processing and communication with the MES / ERP. The old machines are operational, fulfill their production functions, but are not as efficient as desired due to unexpected outages related to failures. In addition do not have the ability to communicate and produce the desired KPIs for the production and maintenance teams. Based on this, the installation of sensors and intelligent controllers (*SmartBoxes*) for data acquisition and processing is essential to allow the Smart Component to perform its functions. The next section details the *SmartBox* that executes the Smart Component.

A first phase of the case study consisted in retrofitting one of the older looms with several sensors in order to gather production data. The set of sensors was selected with the help of the maintenance team, that provided insights for typical breakdown problems and maintenance routines. Figure 4 illustrates the physical architecture for the case study. The Jacquard, which is a complex electromechanic component of the loom, was equipped with: 1) a triple axis accelerometer, connected to the analog pins of the SmartBox, to measure mechanical vibrations; 2) a microphone, connected to an analog pin, to capture mechanical noise; 3) an inductive sensor, connected to a digital pin, facing one of the gears, allowing to count pulses, therefore being able to extract the production speed and quantity; 4) the light tower, connected to three

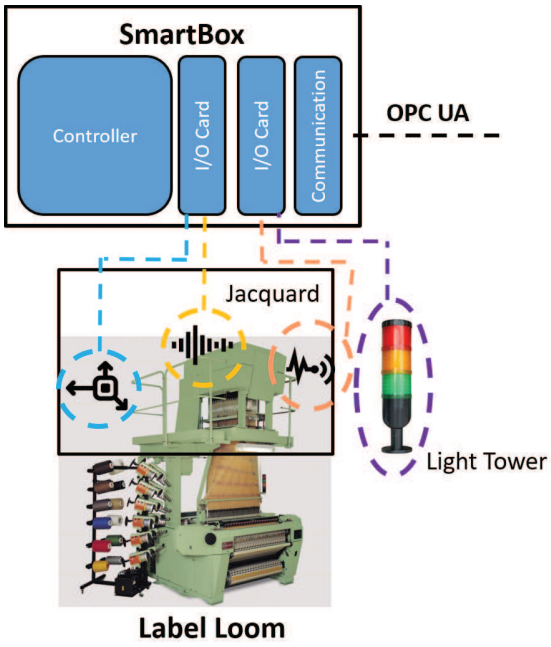

Fig. 4. Physical assets architecture.

digital pins, to detect the machine states, which are encoded by specific pulse sequences.

A. Smart Box

The *SmartBox* is a modular hardware controller developed over a Programmable Automation Controller (PAC) *CompactRIO 9040* from National Instruments [27] that runs the Smart Component at the shop floor level. The CompactRIO system was chosen by the combination of a real-time controller, Reconfigurable IO (RIO) modules and an FPGA module in the same chassis. This combination allows the controller for multitasking in data acquisition, processing and implementation of control algorithms, with a wide range of sampling frequencies. The modularity of the *SmartBox* allows scalability for different industrial scenarios, also allowing the Smart Component to be scalable and adaptable to different realities. Another advantage of using the CompactRIO is that by running a patched Linux Kernel, it allowed to deploy and validate several SCKM aspects. The most important, the data acquisition software is developed in LabView, while the OPC UA server used is implemented in Java. The SCKM allowed to easily interconnect these components to form a data monitoring application.

The *SmartBox* works as an OPC UA server, collects data from sensors, implements data signal conditioning and calibration by software. It has resources to communicate with machine controllers via different industrial communication protocols, converging them into a unified communication architecture enabling link to higher hierarchical levels in the automation pyramid. The Functional block diagram of *SmartBox* is illustrated in Fig. 5.

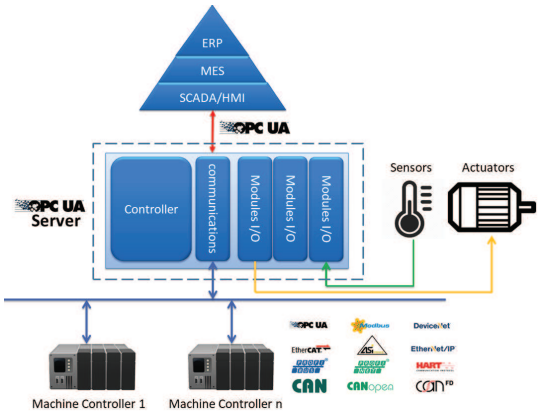

Fig. 5. Functional block diagram of *SmartBox*.

B. Results

In the IDEPA scenario, one *SmartBox* was installed for edge computing of one legacy loom, but is in process the installation of more *SmartBoxes* with capabilities to integrate groups of 4 machines by controller in all factory plant. The connections between machines and the controller are done by remote I/Os under EtherCAT communications.

Older looms are being upgraded with additional devices (rotary encoders and environmental sensors) to enable the collection of operational information already available on new looms. Predictive maintenance functionalities based on information such as temperature, humidity, noise and vibration will also be implemented.

At the time of writing this paper, the SmartBox has been collecting sensor data from the first loom equipped with sensors for roughly 3 months. The plot of Fig. 6 shows an excerpt of sensor data during operation. The Acceleration in X, Y and Z components, Microphone and Lights (Red, Orange and Green) are plotted against the left vertical axis in Volts. The Inductive series is in fact the output of *Impulse to Meters* component depicted in Fig.3. This series is plotted against the right vertical axis in meters and represents the production length.

As can be observed in this excerpt, the loom was operating normally, as the Inductive series indicates, when the Red light started to pulse indicating a breakdown. After this, it is clear that the machine stopped, as can be deduced from the lower peak to peak amplitude of the vibration and acoustic sensors; and also from the flat line of the Inductive series.

V. Future Work

As future work, data from sensors and machine signaling will be used to implement supervised learning algorithms for predictive maintenance and condition monitoring. Datasets of acoustic sensors and accelerometers will be processed by the *SmartBox*, in order to identify and classify the different frequencies, helping decision support systems to identify problems on the looms. One of the main future goals will be to assess how the Smart Component will ease the process of

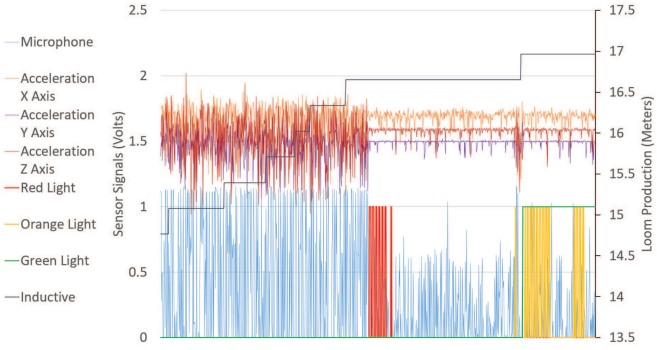

Fig. 6. Operation data an old loom equipped with external sensors.

feeding the algorithms and integrating their predictions with other components.

The currently developed component description techniques do not qualify attributes such as accuracy, availability, latency and security, which are fundamental to deterministic applications. In future iterations of the SOSD language, these characteristics must be revised and incorporated. In the same domain, a real time patch for the Linux Kernel will be included to see how well the SCF handles soft real-time components.

VI. Acknowledgment

This research was supported by the project PRODUTECH-SIF - Solutions for the Industry of the Future, financed by the Portuguese National program COMPETE 2020 and by Portuguese funds through FCT – Fundação para a Ciência e a Tecnologia (UID/EMS/00712/2013).

References

[1] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent manufacturing in the context of industry 4.0: a review," *Engineering*, vol. 3, no. 5, pp. 616–630, 2017.

[2] I. Crnkovic and M. P. H. Larsson, *Building reliable component-based software systems*. Artech House, 2002.

[3] F. Bronsard, D. Bryan, W. Kozaczynski, E. S. Liongosari, J. Q. Ning, Á. Ólafsson, and J. W. Wetterstrand, "Toward software plug-and-play," in *Proceedings of the 1997 Symposium on software reusability*, 1997, pp. 19–29.

[4] G. Gonçalves, J. Reis, R. Pinto, M. Alves, and J. Correia, "A step forward on intelligent factories: A smart sensor-oriented approach," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*. IEEE, 2014, pp. 1–8.

[5] H. Bedenbender, A. Bentkus, U. Epple, T. Hadlich, R. Heidel, O. Hillermeier, M. Hoffmeister, H. Huhle, M. Kiele-Dunsche, H. Koziolek *et al.*, "Industrie 4.0 plug-and-produce for adaptable factories: Example use case definition models and implementation," *Federal Ministry for Economic Affairs and Energy (BMWi), Tech. Rep.*, 2017.

[6] S. K. Panda, T. Schröder, L. Wisniewski, and C. Diedrich, "Plug&produce integration of components into opc ua based data-space," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2018, pp. 1095–1100.

[7] M. A. . C. K. Hoffmeister Festo. Industrie 4.0: The industrie 4.0 component. [retrieved: 6, 2019]. [Online]. Available: \url{https://www.zvei.org/fileadmin/user_upload/Themen/Industrie_4.0/Das_Referenzarchitekturmodell_RAMI_4.0_und_die_Industrie_4.0-Komponente/pdf/ZVEI-Industrie-40-Component-English.pdf}

[8] X. Ye and S. H. Hong, "Toward industry 4.0 components: Insights into and implementation of asset administration shells," *IEEE Industrial Electronics Magazine*, vol. 13, no. 1, pp. 13–25, 2019.

[9] G. T. Heineman and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley*, p. 5, 2001.

[10] F. Fouquet, B. Morin, F. Fleurey, O. Barais, N. Plouzeau, and J.-M. Jezequel, "A dynamic component model for cyber physical systems," in *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, 2012, pp. 135–144.

[11] W. Dai, W. Huang, and V. Vyatkin, "Enabling plug-and-play software components in industrial cyber-physical systems by adopting service-oriented architecture paradigm," in *IECON 2016-42nd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2016, pp. 5253–5258.

[12] Neto, Luis and Gonçalves, Gil, "Component Models for Embedded Systems in Industrial Cyber-Physical Systems," in *The International Symposium on Intelligent Manufacturing Environments InManEnv 2018*, [retrieved: 6, 2019]. [Online]. Available: {https://www.researchgate.net/profile/Luis_Neto8/publication/326693373_Component_Models_for_Embedded_Systems_in_Industrial_Cyber-Physical_Systems.pdf}

[13] H. Bedenbender, M. Billmann, U. Epple, T. Hadlich, M. Hankel, R. Heidel, O. Hillermeier, M. Hoffmeister, H. Huhle, M. Jochem *et al.*, "Examples of the asset administration shell for industrie 4.0 components–basic part," *ZVEI White Paper*, 2017.

[14] L. Rauchhaupt, G. Kadel *et al.*, "Network-based communication for industrie 4.0-proposal for an administration shell," *Federal Ministry for Economic Affairs and Energy (BMWi)*, 2013.

[15] P. e. a. Adolphs, "Structure of the administration shell. continuation of the development of the reference model for the industrie 4.0 component," *ZVEI and VDI, Status Report*, 2016.

[16] G.-G. et al., "Towards a semantic administrative shell for industry 4.0 components," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. IEEE, 2016, pp. 230–237.

[17] F. Burzlaff and C. Bartelt, "I4. 0-device integration: A qualitative analysis of methods and technologies utilized by system integrators: Implications for enginering future industrial internet of things system," in *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2018, pp. 27–34.

[18] ZVEI. (2018) Software als industrie 4.0-komponente. [Online]. Available: https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2018/Oktober/Positionspapier_Software_als_Industrie_4.0-Komponente/Positionspapier_Software_als_Industrie_4.0-Komponente.pdf

[19] A. Widodo and B.-S. Yang, "Support vector machine in machine condition monitoring and fault diagnosis," *Mechanical systems and signal processing*, vol. 21, no. 6, pp. 2560–2574, 2007.

[20] A. Krishnakumari, A. Elayaperumal, M. Saravanan, and C. Arvindan, "Fault diagnostics of spur gear using decision tree and fuzzy classifier," *The International Journal of Advanced Manufacturing Technology*, vol. 89, no. 9-12, pp. 3487–3494, 2017.

[21] J. Reis and G. Gonçalves, "A zero-shot learning application in deep drawing process using hyper-process model," *arXiv preprint arXiv:1901.08969*, 2019.

[22] L. Neto, J. Reis, R. Silva, and G. Gonçalves, "Sensor selcomp, a smart component for the industrial sensor cloud of the future," in *2017 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2017, pp. 1256–1261.

[23] Neto, Luis; Madsen, Anders; Jeppesen, Nicolaj Søndberg; Silva, Ricardo; Reis, João; McIntyre, Peter, and Gonçalves, Gil, "A Component Framework as an Enabler for Industrial Cyber Physical Systems," in *IEEE International Conference on Industrial Cyber-Physical Systems (ICPS 2018)*. [Online]. Available: {https://ieeexplore.ieee.org/abstract/document/8387682/}

[24] O. Foundation, *Specification: Part 5 - Information Model*, 2017, version 1.04. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-5-information-model

[25] L. Neto, G. Gonçalves, P. Torres, R. Dionísio, and S. Malhão, "An industry 4.0 self description information model for software components contained in the administration shell," in *The Eighth International Conference on Intelligent Systems and Applications*. International Academy, Research, and Industry Association, 2019.

[26] "PRODUTECH-SIF - solutions for the industry of the future," http://mobilizadores.produtech.org/en/produtech-sif, [retrieved: 2, 2020].

[27] N. Instruments. (2020) crio9040. [Online]. Available: https://www.ni.com/documentation/en/compactrio-controller/latest/crio-9040/overview/