Information Systems 98 (2021) 101727

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is

The FORA Fog Computing Platform for Industrial IoT☆

Paul Pop [a,*], Bahram Zarrin [a], Mohammadreza Barzegaran [a], Stefan Schulte [b], Sasikumar Punnekkat [c], Jan Ruh [d], Wilfried Steiner [d]

[a] DTU Compute, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark
[b] Distributed Systems Group, Vienna University of Technology, Karlsplatz 13, 1040 Wien, Austria
[c] Dependable Software Engineering, Mälardalen University, Högskoleplan 1, 722 20 Västerås, Sweden
[d] TTTech Labs, TTTech Computertechnik AG, Schoenbrunner Strasse 7, 1040 Vienna, Austria

ARTICLE INFO

Article history:
Received 1 July 2020
Received in revised form 17 November 2020
Accepted 24 January 2021
Available online 2 February 2021
Recommended by Gottfried Vossen

Keywords:
Fog Computing
Industrial IoT
Industry 4.0
AADL
Time-Sensitive Networking
Deterministic virtualization

ABSTRACT

Industry 4.0 will only become a reality through the convergence of Operational and Information Technologies (OT & IT), which use different computation and communication technologies. Cloud Computing cannot be used for OT involving industrial applications, since it cannot guarantee stringent non-functional requirements, e.g., dependability, trustworthiness and timeliness. Instead, a new computing paradigm, called *Fog Computing*, is envisioned as an architectural means to realize the IT/OT convergence. In this paper we propose a Fog Computing Platform (FCP) reference architecture targeting Industrial IoT applications. The FCP is based on: deterministic virtualization that reduces the effort required for safety and security assurance; middleware for supporting both critical control and dynamic Fog applications; deterministic networking and interoperability, using open standards such as IEEE 802.1 Time-Sensitive Networking (TSN) and OPC Unified Architecture (OPC UA); mechanisms for resource management and orchestration; and services for security, fault tolerance and distributed machine learning. We propose a methodology for the definition and the evaluation of the reference architecture. We use the Architecture Analysis Design Language (AADL) to model the FCP reference architecture, and a set of industrial use cases to evaluate its suitability for the Industrial IoT area.

1. Introduction

We are at the beginning of a new industrial revolution (Industry 4.0), which will bring increased productivity and flexibility, mass customization, reduced time-to-market, improved product quality, innovations and new business models. Industrial Internet of Things (IIoT, also called Industrial Internet) is a key enabling technology for Industry 4.0, where the focus is on interconnected machines [1]. IIoT is providing the infrastructure that underpins our Smart Society (Smart Energy Grid, Smart Cities, Smart and Green Mobility, Smart Manufacturing, etc.), providing solutions for several societal challenges.

However, Industry 4.0 will only become a reality through the convergence of Operational and Information Technologies (OT & IT), which are currently separated in a hierarchical pyramid (Purdue Reference Model [2], see Fig. 1) and use different computation and communication technologies. OT consists of cyber–physical systems that monitor and control physical processes that manage, e.g., automated manufacturing, critical infrastructures, smart buildings and smart cities. These application areas are typically safety-critical and real-time, requiring guaranteed extra-functional properties, such as, real-time behavior, reliability, availability, safety, and security and often required to show compliance to industry specific standards. OT uses proprietary solutions implemented with barriers between each level in the pyramid in Fig. 1, imposing severe restrictions on the information flow.

Instead, a new paradigm, called Fog Computing, is envisioned as an architectural means to realize the IT/OT convergence in Industrial IoT [3]. *Fog Computing* is a "system-level architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things" [4]. With Fog Computing, communication devices, such as switches and routers are extended with computational and storage resources to enable a variety of communication and computation options. Fog Computing will enable a powerful convergence, unification and standardization at the networking, security, data, computing, and control levels. It will lead to improved interoperability, security, more efficient and rich control, and higher

☆ The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785.
* Corresponding author.
E-mail addresses: paupo@dtu.dk (P. Pop), baza@dtu.dk (B. Zarrin), mohba@dtu.dk (M. Barzegaran), s.schulte@infosys.tuwien.ac.at (S. Schulte), sasikumar.punnekkat@mdh.se (S. Punnekkat), jan.ruh@tttech.com (J. Ruh), wilfried.steiner@tttech.com (W. Steiner).

https://doi.org/10.1016/j.is.2021.101727
0306-4379/

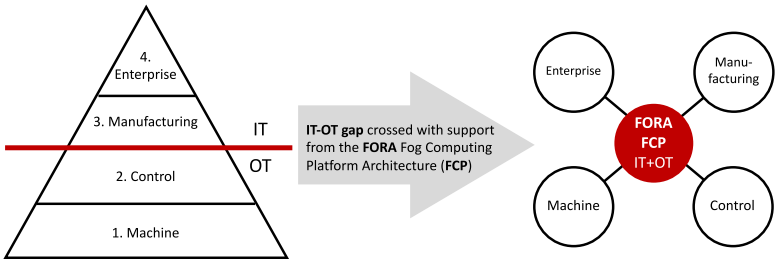


Fig. 1. Industry 4.0: IT/OT convergence supported by the FORA Fog Computing Platform (FCP).

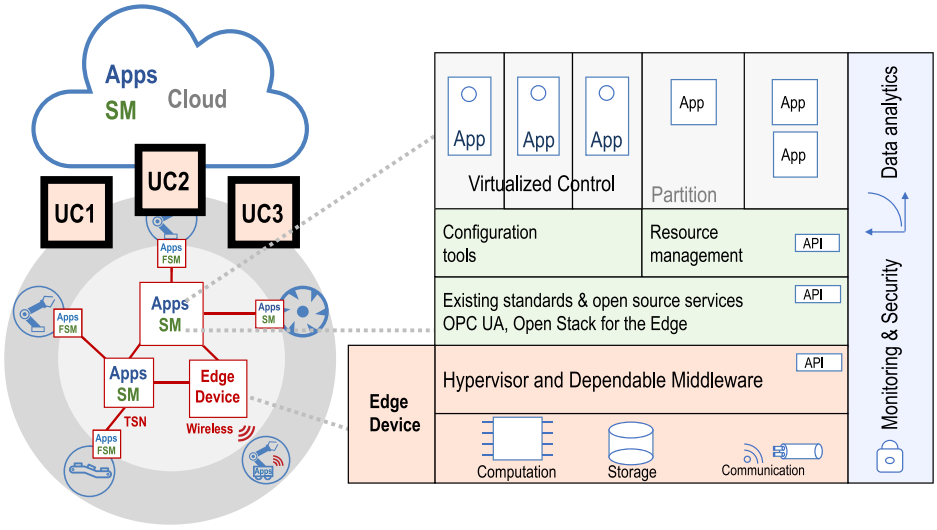


Fig. 2. FORA Fog Computing Platform concept: UCs and main components.

manufacturing efficiency and flexibility [5]. Several initiatives are currently working towards realizing this vision [6,7].

One notable initiative is the European Training Network on Fog Computing for Robotics and Industrial Automation (FORA ETN[1]). The vision is to virtualize the industrial control (which is then implemented as control applications running on a Fog Computing Platform) and achieve the same level of dependability as the one taken for granted in OT. The convergence of IT and OT will be supported by: the increased usage of IP-protocols, e.g., standardized Deterministic Ethernet solutions from IEEE Time-Sensitive Networking (TSN) Task Group [8], upcoming 5G wireless standards [9], and interoperability standards such as OPC Unified Architecture (OPC UA) [10], all integrated into a Fog Computing Platform (FCP), (see Fig. 2), which brings computation, communication and storage closer to the edge of the network. FORA's research objectives focus on: future industrial automation architectures and applications based on Fog Computing, deterministic virtualization and execution, deterministic wired and wireless communication, resource provisioning and resource management, service-oriented architecture solutions, real-time data analytics and security.

1.1. Contributions

In this paper we propose a *reference system architecture* for a safe and secure *Fog Computing Platform* (FCP). In our context, an *architecture* captures the fundamental organization of a system (a collection of components performing a specific set of functions) in terms of its components, their relationships to each other, and to the environment, and the principles guiding its design and engineering. A *reference* architecture provides a proven template solution for an architecture for a particular domain; in our case, we are targeting the Industrial IoT area. A Fog Computing *Platform* provides a set of technologies that are used as a base upon which mixed-criticality applications (safety critical, industrial, fog applications) are developed. The proposed reference platform architecture uses open standards and open source, e.g., TSN, OPC UA, 5G and OpenStack for the Edge.[2]

We propose a methodology to derive and evaluate the proposed FCP architecture. We use the standardized Architecture Analysis Design Language (AADL) [11] (which is an SAE standard) to model the FCP architecture. We have used several industrial Use Cases (UCs) [12–14] to evaluate the proposed architecture. We report in the paper in detail the results obtained by using the architecture to implement an industrial conveyor distribution system UC.

The related work is presented in Section 2. Section 3 introduces the process that has been used to derive the FORA Fog Computing Platform Architecture. The details of the Fog Computing architecture that we propose for Industrial IoT are presented in Section 4 using AADL. We discuss the evaluation of the architecture in Section 5 where we focus on an industrial use case. Finally, we conclude the paper and provide a discussion in Section 6.

2. Related work

Within this section, we discuss the most important related work relevant to the FORA Reference Architecture. Especially, we

[1] See the FORA project website for more details, http://fora-etn.eu.

[2] https://wiki.openstack.org/wiki/Edge_Computing_Group.

will have a look at reference architectures in the field of fog computing (Section 2.1), the manifold approaches to resource management and optimization that have been presented in recent years (Section 2.2), and will discuss the utilization of fog computing in industrial settings (Section 2.3).

2.1. Reference architectures

Especially in the first years of research on fog computing, a number of (reference) architectures have been proposed, starting with the seminal work on fog computing by Bonomi et al. [3,5]. There, the authors have already presented a layered model, defining fog computing on a high level. The different layers include computational resources in embedded sensors, multi-service edge, the core network and (cloud-based) data centers. An early reference architecture has been presented by Dastjerdi et al. [15], where the authors also divide the fog into a number of hierarchical layers, including IoT devices at the edge of the network, the network itself, cloud services and resources, software-defined resource management, and IoT applications running on top of the fog resources.

A standardized reference architecture for fog computing is the "OpenFog Reference Architecture for Fog Computing" [4] proposed by the OpenFog Consortium, which has later merged with the Industrial Internet Consortium (IIC). This reference architecture has in 2018 been standardized by IEEE as the "1934–2018 IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing" [16]. The reference architecture is quite extensive, covering functional pillars of fog computing, a number of use cases for fog computing, as well as the actual reference architecture, provided from different viewpoints. Since the OpenFog Reference Architecture is quite versatile, it does naturally not cover details for industrial settings, which is the focus of the FORA reference architecture. Another important initiative towards standardization in the field of fog and edge computing is led by the European Telecommunications Standards Institute (ETSI), and focusing on Multi-Access Edge Computing (MEC) [17]. Notably, the ETSI activities are very broad and provide a large number of publications which provide a lot of details, up to the level of providing concrete API specifications. In contrast to the work at hand, the ETSI MEC activities rather focus on the edge level, however also taking into account fog aspects. Also, the ETSI MEC activities are use case-agnostic, and therefore not specifically aiming at industry settings, as we take into account.

Apart from the two major reference architectures by the OpenFog Consortium and ETSI, there are a number of more specific reference architectures which are not backed by a large consortium or standardization association: Puliafito et al. [18] present a reference architecture for a "follow-me fog", which dynamically migrates services based on user mobility. Habibi et al. [19] focus more on the communication aspects by integrating software-defined networking into a fog computing reference architecture. De Brito et al. [20] discuss an extension of the OpenFog Reference Architecture by providing the means for service orchestration.

There are also reference architectures aiming at specific use cases: Mahmud et al. [21] extend basic architecture models for fog computing by adding specific components needed for smart healthcare scenarios, e.g., facilitating interoperability between existing fog clusters. Qi et al. propose a reference architecture for smart manufacturing applications [22]. Here, the authors address an application area comparable to the work at hand. Notably, the reference architecture considers that there is a need for real-time capabilities in fog computing, and that specific resources need to be utilized to achieve this. Their reference architecture contains the means for control, interaction, information integration, and collaboration in smart manufacturing settings. However, in contrast to our work, there is no specific discussion of hard real-time constraints or control applications.

In contrast to the reference architectures discussed so far, our work explicitly aims at industrial settings, taking into account hard real-time constraints and specific needs of control applications.

Apart from the already discussed reference architectures, there have been further discussions on how to structure fog architectures. Mostly, current fog systems make use of a hierarchical architecture, as has already been proposed in the OpenFog Reference Architecture, e.g., [23,24]. However, there are also architectures which apply a flat structure and apply basic principles from the field of self-organization and peer-to-peer computing, e.g., [25,26]. While hierarchical approaches are the current state-of-the-art, they possess some potential drawbacks, e.g., that some nodes within a hierarchical fog system may become bottlenecks or single points of failure [27]. Therefore, within the proposed FORA Reference Architecture, we apply a flat, fully distributed architectural style.

2.2. Resource management

Resource management has been a very popular research topic in the field of fog computing in the last few years, with many approaches having been presented so far [28,29]. The applied methods to manage fog resources are manifold, and range from optimal solutions, e.g., applying mixed-integer linear programming [30], to heuristics, e.g., applying Genetic Algorithms [31]. While most presented solutions are use case-agnostic, there are also specific approaches for vehicular fog computing [32], smart healthcare [33], image processing [34], or industrial settings [35,36]. According to a systematic literature review by Bellendorf and Mann [28], most approaches to resource management aim at optimizing latency, while energy efficiency also plays a major role. In contrast, cost efficiency, which has been a major research topic in the field of cloud computing [37], is not so prominently discussed in fog computing.

As outlined in the work at hand, hard real-time behavior and safety-related aspects are indispensable in the Industrial IoT. Despite the fact that latency is a major topic in fog resource management, very few studies explicitly aim at hard real-time behavior. For instance, Raagard et al. [38,39] discuss the runtime reconfiguration of TSN schedules for fog computing, thus supporting applications composed out of hard real-time control tasks. Fizza et al. [40] take into account real-time capabilities during scheduling of tasks and services in the fog. For this, the Earliest Deadline First approach is applied. A similar approach is provided by Gomes et al. [41], aiming at healthcare environments. A conceptual approach to enable real-time fog computing is presented by Kopetz and Poledna [42], where the authors discuss the basic concept of time-triggered VMs. In addition, the support of soft real-time applications is mentioned in quite a large number of studies. For instance, Ning et al. discuss the utilization of fog computing to support real-time traffic management for smart cities [43], and Verma and Sood discuss real-time capabilities in smart healthcare scenarios [44].

2.3. Fog computing in industrial settings

We have already discussed some approaches to apply fog computing in industrial settings in the former subsections, focusing on (reference) architectures and resource management for industrial settings, respectively. Within this subsection, we will further discuss related work in this area.

As pointed out above, one important trend in the Industrial IoT is the convergence of OT & IT. Fog computing has been named as an architectural means to achieve this convergence [45,46]. More concretely, Müller et al. [47,48] present a reference model for a seamless runtime environment for industrial software, which can thus be deployed in both the fog and the cloud. This model is implemented by orchestrating containers running the single applications. Similarly, Meixner et al. [49] present a framework for automatically placing applications in fog environments, especially aiming at industrial settings.

Apart from this, a number of studies present how to deploy functionalities, which are usually hosted in centralized data centers, closer to the industrial resources, e.g., Cyber–Physical Production Systems (CPPS), by making use of fog resources. Aazam et al. [50] especially discuss the specific requirements of CPPS, and how fog computing architectures can help to overcome their challenges, with a focus on communication, CPS control, (big) data analysis, and sensing. Colelli et al. [51] show how fog resources can be used to improve security in OT and IT networks. Li et al. [52] present an approach to make use of machine learning in the fog in order to improve machine maintenance. Similar work is presented by O'donovan et al. [53]. Fernández-Caramés et al. [54] utilize fog resources in order to enable augmented reality in industrial settings. Zhou et al. [55] discuss how to control CNC machine tools through a fog-based solution. The goal here is once again to achieve (soft) real-time control of CPS in Industrial IoT settings.

While we have only discussed a very limited number of possible applications for fog computing in the Industrial IoT, it can be easily seen that the potential applications cover a very broad spectrum of functionalities. The presented applications could be integrated into the FORA Reference Architecture, thus exploiting the control and real-time capabilities of our work.

3. Reference platform architecture definition and evaluation methodology

This section introduces briefly the FORA project and proposes a methodology for the reference platform architecture definition and evaluation. FORA is a four-year European Training Network, started in 2017, with the aim of developing a Fog Computing Platform for Industrial IoT. FORA also trains 15 Ph.D. candidates at 7 partner organizations (both academic and industrial), involving also 5 other companies that provide hosting, training and use cases. FORA has a team of over 50 researchers; each Ph.D. candidate has three advisors, both from industry and academia.

Fig. 2 presents a high-level conceptual overview of the FORA Fog Computing Platform architecture. In the left part of the figure, boxes with a red border represent fog nodes, connected with each other and to the Cloud; the thick lines in-between these boxes are the network. Applications (Apps) run in the Fog and Cloud. The FORA project is organized around three main research themes: Fog Computing Platform, which virtualizes computation, communication and storage; Resource Management and Middleware, which uses the platform to provide guarantees for the industrial control applications and novel Fog/Cloud resource management mechanisms (via a Software Manager—SM); and Dependability Services and Application Modeling, which are vertical services to ensure safety/security aspects and horizontal services to unlock high-value data analytics, implemented as Fog applications. The main building blocks of the platform architecture will be presented in detail in Section 4. Here, we discuss the steps of the methodology used for defining the platform architecture, and how the resulted architecture has been evaluated. The details of the evaluation are covered in Section 5.

We have identified within FORA several Use Cases (UCs) that are relevant to the FORA organizations and which have a good coverage of the Industrial IoT area: UC1—Electric drives as Fog Nodes in a industrial setting; UC2—Fog-based Industrial Robotic System; UC3—Next generation of machine control using an Edge Platform. A high-level description of these UCs is presented in [56].

The UCs drive the identification of requirements, Key Performance Indicators (KPIs) and UC-specific evaluation metrics. The requirements provide the specific constraints and problems that have to be addressed in the research themes. The evaluation metrics and KPIs establish goals that have to be achieved, and that can be assessed in the evaluation. We have organized the FORA researchers into three teams, one per use case. These use case teams have been asked to identify requirements, KPIs and evaluation metrics. We have provided a template to collect the requirements, consisting of: Requirement ID, Description, Rationale, Abstraction level, AADL component names, and relevance to which research theme.

We have collected 10 KPIs and about 80 requirements, see [56] for details. We have consolidated these 80 initial requirements, based on feedback from all the stakeholders, into a coherent set of 47 requirements. The requirements-related documents have been periodically updated during the project based on feedback from the research work and use cases. A partial list of these requirements is presented in Table 1, where we have eliminated the list of the UC-specific and low-level requirements due to space limitations.

We have decided to use the Architecture Analysis and Design Language (AADL) [11] to model the FORA FCP architecture, see Section 3.1. The FORA AADL models are shared via a "git" repository with all the partners, which contribute to it. The researchers working in each research theme have been tasked with the definition of its respective platform building blocks and the related AADL models, see the sub-sections of Section 4.

To evaluate the proposed reference platform architecture, we have: (i) asked the FORA researchers and other stakeholders, to provide feedback on the AADL models that define the reference platform architecture; (ii) used the AADL models to model the three mentioned use cases; and (iii) implemented the use cases as "demonstrator prototypes" to evaluate the ability of our platform architecture to support the design and engineering of IIoT systems. The concrete research outputs of the FORA project, which can be a hardware or software prototype, a method, a tool, a model, etc. are gathered as a set of "Technology Bricks" (TBs). Thus, for (iii), we have used the AADL reference architecture meta-model to model the UCs and integrated these TBs into the demonstrators, one for each UC. We have focused on achieving a high-level of coverage of requirements in the AADL architecture meta-model and a high coverage of AADL components and TBs used in the demonstrators. Section 5 presents the evaluation results focusing on the evaluation via use cases, namely using the conveyor distribution system demonstrator built for UC1.

3.1. Architecture Analysis and Design Language

The Architecture Analysis and Design Language (AADL) is a well-known architecture description language in the domain of real-time embedded systems, which has been introduced by the Society of Automotive Engineers (SAE) [57]. It can model both software and hardware components of a system in a modular and component-oriented approach.

Unlike other modeling languages, e.g., UML and SysML, AADL provides both textual syntax and graphical notations with precise semantics. It introduces a different category of components to model a system, including software components (e.g., data, thread, thread group, subprogram, process), hardware components (e.g., memory, bus, processor, device). It also provides hybrid components (abstract, system) to allow hierarchical system

Table 1
Selected requirements for the FORA Fog Computing Platform.

ID	Description	Rationale
R1	The hardware platform shall provide support for virtualization	Necessary to allow the use and application of modern available hypervisors
R2	The platform shall provide temporal and spatial isolation via hypervisors	Enables hosting of mixed-criticality applications: the lower criticality functions should not impair the safety of the higher criticality functions
R3	The platform shall support lightweight container-based virtualization	This helps to avoid the deployment overhead introduced by the use of hypervisor-based virtualization
R4	The platform shall provide deterministic inter-node communication using standard protocols	To guarantee bounded latency communication between the fog node and its environment, enabling the re-location of critical applications from the machine to the fog node
R5	The platform shall provide reliable and timely wireless communication for the mobile fog nodes	To meet the strict requirements on reliability and latency communication of mobile fog applications such as mobile robots
R6	The platform shall provide fault tolerant communication among the participating compute nodes	In case one or more of the participating compute nodes become unresponsive, the platform should be able to maintain the connectivity of the responsive compute nodes
R7	The platform shall provide standards-based middleware for both industrial and Fog applications	To support the development and deployment of mixed-criticality applications
R8	Each fog node shall broadcast its hardware capabilities	To ensure efficient resource management, any resource manager shall know the available hardware resources within the fog network
R9	The platform shall support the specification and enforcement of security policies	Security policies describe what kind of actions are permissible in a network, and are valuable tools in the prevention and containment of malicious activity
R10	The fog node shall be able to detect errors during its operation and to recover	To ensure the fault-tolerant and high-integrity operation of the safety-related applications
R11	The platform shall be able to run critical control applications	This is needed to virtualize the control equipment (such as PLCs) onto the platform to reduce hardware costs and increase flexibility
R12	The platform shall be compliant with POSIX standards whenever applicable	To ensure portability of applications
R13	Critical real-time tasks should inform their worst-case execution time, period, deadline	To enable the allocation of the necessary resources to the critical tasks to meet their deadlines
R14	The fog nodes shall be able to run data analytics at the edge	To avoid sending all data to the cloud and to support fast optimization and better FCP resource utilization
R15	The platform shall allow for secure retrieval, verification, and execution of software updates	The ability to update has proven itself to be a critical component in the continuous effort to build secure systems

composition and model extension and refinement through the design process.

Similar to object-oriented programming, components can be defined in two levels; component types and component implementations. Component types define the interface of a component, including its external features, e.g., input and output ports. While component implementation specifies the internal elements of a component, such as sub-components and their interactions through connections. Both component types and component implementations can extend other component types or implementations, and each component type can have zero or multiple implementations. Component definitions should be structured within AADL Packages to declare a namespace for them.

Furthermore, AADL allows attaching typed values to the components via properties to specify constraints or characteristics concerning the architecture elements. Several standard properties are available such as timing properties for threads, MIPS capacity for the processors, bandwidth capacity for bus components. Most of AADL analysis tools understand and use these standard properties. It is also possible to define a new set of properties through AADL Property Sets. We use this extensibility mechanism to introduce new properties for specification and configuration of time criticality applications, fog nodes, TSN networks and switches in the FORA reference architecture model.

There are several tools developed for the AADL language to facilitate modeling and analysis of embedded systems from different perspectives such as real-time performance, resource consumption, security, etc. The most well-known one is OSATE [58], which is an open-source Eclipse-based modeling framework. In addition to the modeling environment for the AADL language, it provides a set of plug-ins for validating and analyzing the architecture of the system under study. We have chosen to use AADL as the core language for modeling FORA FCP reference architecture due to its non-ambiguous semantics, human readability, extensibility, and availability of a large set of analysis tools, e.g., scheduler, model checker, flow latency analysis, etc., as OSATE plug-ins.

We have developed the FORA AADLs models based on the standard AADLs components; we have also extended previously proposed meta-models. For example, we have extended the ARINC653 module [59], an AADL annex, which defines virtual processor and virtual buses and targets safety-critical real-time systems, to model hypervisors and address the virtualization and partitioning requirements of the FORA FCP.

As mentioned earlier, FORA has developed a set of Technology Bricks, including a set of methods and tools, e.g., to configure different elements of the FORA FCP platform, such as network topology and routing design [60], streams and task scheduling [61]. We integrate these tools as plug-ins into the OSATE modeling environment in order to facilitate the platform configuration and validation of the FCP use cases. In addition, the models can be analyzed through a set of OSATE-specific plug-ins developed for analyzing safety-critical real-time systems and can be transferred to set up the configuration of the target systems thanks to the Ecore code generation and model transformation mechanisms [62].

Let us illustrate the main concepts of AADL using a simple system consisting of a sensor, an actuator, a computation platform, and an application that consists of two critical control tasks. Fig. 3 presents the architecture of this system as an AADL system component that contains all the system's component instances. This system component represents the top-most level system, which provides the root of the architecture tree, and it must be instantiated to conduct architecture analysis. The sensor and the actuator are modeled as two individual systems with one port to send or receive data. The application is modeled via two

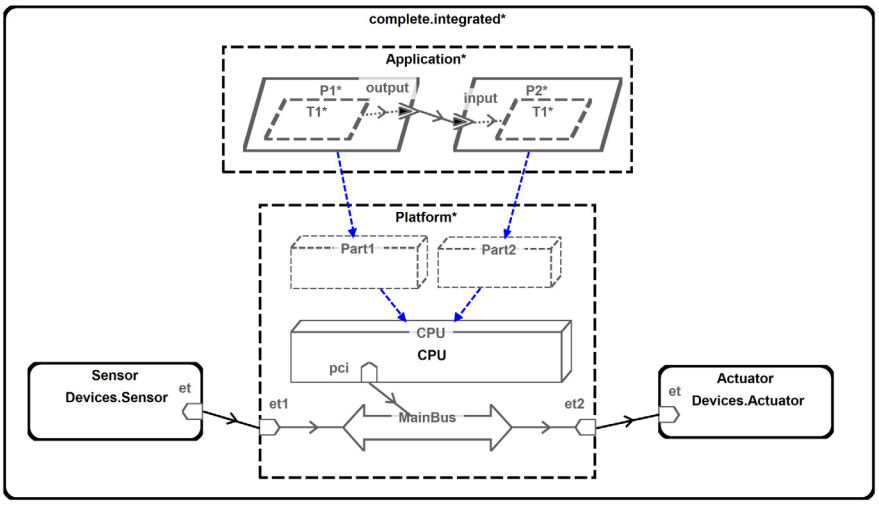


Fig. 3. An illustration of the main AADL modeling concepts used in this paper. System components are: An FCP (shown with "Platfrom"), A sensor (shown with "Devices.Sensor"), and an actuator (shown with "Devices.Actuator").

processes, which each host one thread; P1 generates data, and P2 consumes the data. Thread components model the active part of an application, and they should be contained in the process components which model the address spaces that contain the threads. We model the platform as an abstract system with two ports for the connectivity purpose to the sensor and actuator, one processor, and a bus to connect them. Abstract components are partially defined components that can be refined during the modeling process. They are especially useful when defining a model as a reference architecture. We choose to use partitioning in order to isolate the execution of the critical tasks on the Processor. Therefore, we model the partitions via two virtual processors introduced by the ARINC653 extension to model a dedicated scheduling domain inside a processor. We use AADL actual processor binding, depicted via blue arrows, to map the processes to the partitions and the partitions to the Processor. For simplicity, we hide the binding visualizations from the AADL diagrams presented in this paper.

4. FORA Fog Computing Platform reference architecture model

The proposed FORA FCP architecture is described in the following three sections from three perspectives: (1) fog computing devices (fog nodes) and the communication among fog nodes, see Section 4.1, (2) the mechanisms and techniques for resource management, orchestration and configuration, see Section 4.2 and (3) services for dependability, analytics and security, see Section 4.3.

An overview of the high-level conceptual architecture is shown in Fig. 4. The foundation of the FCP is the *Fog Node* (FN). In many applications, including industrial automation and robotics, several layers of FNs with differing computation, communication and storage capabilities will evolve, from powerful high-end FNs to low-end FNs with limited resources. Researchers have started to propose solutions for the implementation of FNs [5,6] and fog node solutions have started to be developed by companies [6, 7,63]. In our case, FN is equipped with a Commercial Of-The-Shelf (COTS) multicore processor (MCP), accelerators, such as FPGAs, for machine learning, and advanced wired and wireless networking capabilities. The FN utilizes its advanced networking capabilities to interact with its environment that includes sensors, actuators, other FNs, and remote Cloud facilities. The initial goal of the IEEE 802.1 TSN Task Group [8] was to provide timing guarantees for demanding applications such as those in the automotive area. Thus, IEEE 802.1 TSN is the ideal technology choice for the fog node's southbound connection. A TSN switch can be part of the fog node, as is the case in the automotive area, or may be a separate device, as is typically the case in the industrial area. The vision with TSN is to provide a superior technical solution based on open standards. TSN guarantees bounded latency communication between the fog node and its environment. This guarantee enables the re-location of real-time critical tasks from the machine to the fog node. Furthermore, industrial wireless technologies, e.g., WirelessHART or 5G, enable communication with mobile entities or Cloud Facilities in case of remote FN installations, e.g., on offshore oil rigs.

When mixed-criticality functions share the same MCP, they are separated in different virtual machines (partitions) enforced using hardware-supported virtualization [64], based on hypervisors, such as PikeOS [65], ACRN or Xen. The FCP hosts a diverse set of applications with mixed-criticality requirements belonging to both OT and IT domains. The applications are distributed over multiple resources, e.g., low-end FNs integrated in machines, high-end FNs hosting multiple applications of mixed-criticalities, or in the Cloud. This is realized through: A runtime environment and means to orchestrate different applications; cross-layer resource allocation, allocating resources efficiently in volatile scenarios, taking into account different types of resources and a number of non-functional requirements, e.g., latency, cost, security, sensitivity of data; configuration mechanisms and tools, which provision resources such that industrial applications meet the dependability requirements, functioning correctly even in the presence of faults, requiring hence dynamic reconfiguration of computation and communication resources. We build on existing open source software stacks for the edge, e.g., OpenStack for the Edge. The Fog middleware will also build on application layer protocols such as MQTT-SN [66] and CoAP [67] for northbound communication and TSN and OPC UA for southbound communication.

We model the FORA FCP elements in a multi-layered approach, which consists of four layers, namely, Core, Software, Hardware, and Platform. The core layer represents the elements at a very abstract level. The Software and Hardware levels enrich the Core layer elements with the software and the hardware components. Finally, the Platform level composes and encapsulates the software and hardware components defined for each element at the Core layer into a single component that can be used to model a Fog Computing Platform.

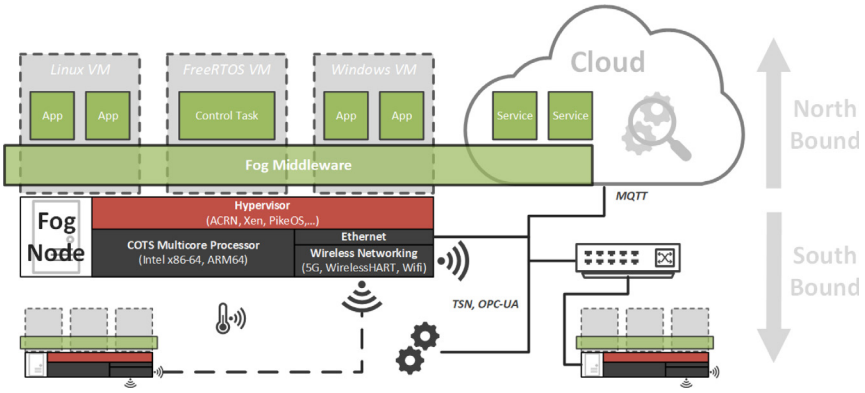


Fig. 4. An illustration of the main components of the FORA Fog Computing Platform.

4.1. Fog node and communication

The FN envisioned in FORA comes with an Intel x86-64 or ARM64 COTS MCP that implements hardware virtualization extensions, such as Intel's VT-x and VT-d, Second Level Address Translation (SLAT), and Single-root I/O Virtualization (SR-IOV). Hardware virtualization extensions allow the hypervisor to host unmodified guest operating systems in VMs. Therefore, Intel VT-x (and its equivalent on ARM64) introduces an additional mode of operation with highest privileges to the instruction set architecture (ISA) that allows native execution of most sensitive instructions [68]. Intel's VT-d (and equivalent extensions on ARM) enable device passthrough that is granting a VM exclusive secure access to a PCI-e device. SR-IOV brings hardware support for sharing a physical PCI-e device with multiple VMs by offering so-called virtual functions. A chipset and PCI-e device with SR-IOV support allows sharing, e.g., a network card with multiple VMs, whilst guaranteeing isolation between the VMs. We do not require each FN having all of the just mentioned hardware extensions and features. The set of required features highly depends on its applications. Therefore, we differentiate three classes of FNs, which together make up the basis for the Fog Computing Platform. Note that the boundaries between the classes and their requirements are fluid:

- Class-1: FNs operating in very close proximity to machines and robots consisting of a multitude of sensors and actuators. The FN takes over critical hard real-time control tasks running in real-time VMs, hence it utilizes suitable COTS MCPs that come with less computational power yet a higher degree of determinism. Furthermore, the FN deploys a statically partitioned hypervisor, such as PikeOS [65], in order to fulfill strict timing requirements of its VMs. Southbound, the FN communicates via typical industrial field busses and/or TSN with OPC UA for non-critical communication. Northbound, thus in connection to other FNs, we leverage traditional Ethernet and TSN with machine-to-machine (M2M) protocols such as OPC UA and the ISO standard Message Queuing Telemetry Transport (MQTT).
- Class-2: FNs operating on the factory floor level. The FN does not take over critical control tasks, yet it can run soft real-time tasks, such as non-critical control, real-time data acquisition, data analysis, and data pre-processing. Therefore, the FN comes with a more powerful COTS MCP and a high degree of connectivity, including wired as well as wireless means of communication. The FN's hypervisor must be more dynamically configurable, such as ACRN or Xen, in order to be able to adapt to changes on the factory floor during runtime. However, soft real-time tasks might still be placed in statically configured VMs. An example for a class-2 FN is the Nerve MFN100 product from TTTech Computertechnik AG running the Xen hypervisor [63].
- Class-3: FNs operating on the factory or enterprise level. The FN collects operational data from all entities on the factory floors, either for sophisticated data analysis and process optimization, for long-term storage, or for forwarding data to the cloud. This requires a high degree of computational power and network throughput as provided by typical server processors and Gigabit Ethernet or even optical fiber since the FN effectively acts as cloud gateway. Southbound communication still involves TSN and OPC UA whereas northbound communicates is solely based upon TCP/IP and MQTT. The hypervisor must manage the FN's hardware resources dynamically and if need be even allow for over-provisioning of resources. Furthermore, integration with cloud services must be straightforward. Therefore, we use hypervisors that can usually be found in typical cloud environments, such as Xen or KVM.

Each FN utilizes hypervisor technology. A hypervisor is a low-level software layer that provides the abstraction of VMs to operating systems. A VM is a set of virtual resources such as, virtual CPUs, main memory, virtual I/O devices, or virtual time. The hypervisor manages the mapping of virtual to physical resources of all VMs it is hosting while guaranteeing strict isolation between its VMs. This allows a hypervisor to partition its hardware and run mixed critical applications isolated in dedicated VMs. Isolation does not come for free: State-of-the-art hypervisors for hard real-time applications, henceforth referred to as class-1 hypervisors, statically partition their resources, such as processor cores and time, main memory, and I/O devices, before runtime in order to achieve strict temporal and spatial isolation between VMs. As a result of their static configuration, class-1 hypervisors lack the flexibility to add, remove, and migrate VMs during runtime which make them less suited for more dynamic Industry 4.0 use cases. To that end we introduce a second class of hypervisors, namely class-2 hypervisors, that provide a good balance of temporal and spatial isolation and flexibility by utilizing mode changes [69] and compositional scheduling theory for the analysis of hierarchical scheduling [70], that is the scheduling of real-time tasks on virtual CPUs that in turn are being scheduled on physical cores of a MCP. Finally, there are FNs that take over high-level management tasks that require a high degree of flexibility. Therefore, they must be able to dynamically create VMs, migrate, and destroy VMs depending on the current task sets and their respective processing demand. We refer to these as class-3 hypervisors.

Furthermore, there are concepts that have to be considered for all three classes of hypervisors, such as the notion of a global time base that requires precise clock synchronization of FNs including VMs and hypervisors.

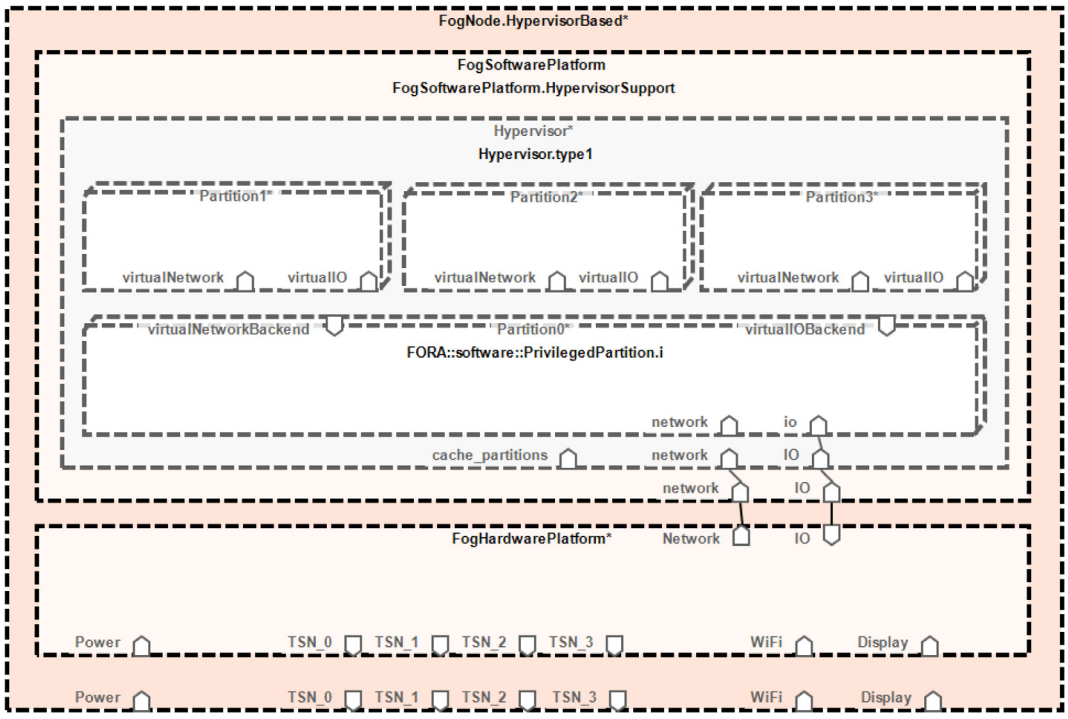


Fig. 5. FORA fog node design overview.

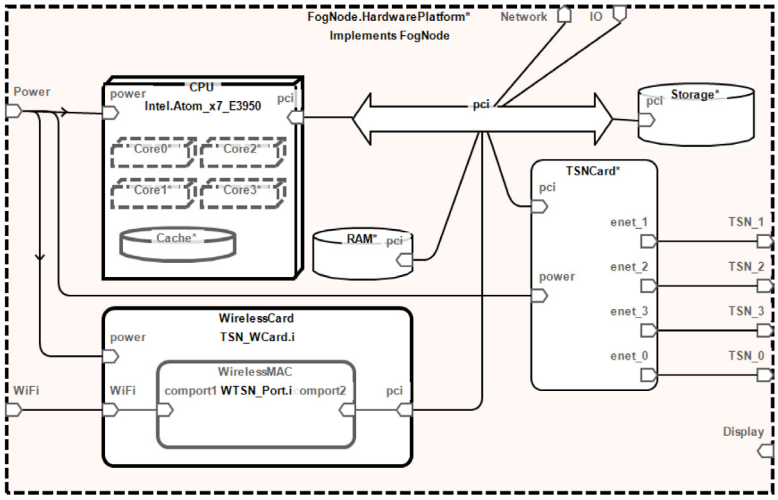


Fig. 6. FORA fog node hardware architecture.

4.1.1. AADL models

We have modeled the components discussed in the previous section using AADL. The FCP is composed of a FN hardware platform and a virtualization solution, which can support both hypervisors and containers. The FN hardware platform has to be deployable in a wide range of industrial scenarios and support the execution of a variety of real-time system classes with different timing requirements [68]. An overview of the FN's design is presented in Fig. 5 and an illustration of the proposed FN's hardware architecture using AADL is presented in Fig. 6.

As mentioned, we utilize a hypervisor to guarantee temporal and spatial partitioning of the fog node hardware platform, see "Hypervisor" component in Fig. 5. The hypervisor provides virtual machines or partitions (see "Partition1", "Partition2", and "Partition3" components in Fig. 5) each of which runs their own operating system (OS). A privileged partition (see "Partition0" component in Fig. 5) per fog node is in control of multiplexing access to shared devices by providing virtual devices. A virtual device consists of a frontend and a backend component whereas the frontend component, e.g., the virtualNetwork component, runs in an unprivileged partition and communicates with the backend component, e.g., the virtualNetworkBackend, in the privileged partition. We provide virtual devices for the I/O interface and the network interface. They connect via PCI to the fieldbus or the internal TSN switch of the FN hardware platform, as shown in Fig. 6.

The processor of the platform is envisioned to be an interchangeable component that can feature either a COTS MCP such as the Intel Atom shown in the CPU component in Fig. 6 or a more specialized real-time multi-core system able to provide bandwidth guaranteed core-to-core communication, time-predictability and static worst-case execution time analysis [69].

A fundamental aspect of Fog Computing is networking, that is why the proposed FN features three communication interfaces.

For the communication paradigm, we propose the usage of TSN as it has been identified as the upcoming standard for real-time communication in Ethernet networks. It allows for mixed-criticality bounded latency communication, by separating traffic into priority classes. The arbitration of the traffic classes operates according to a schedule. To synchronize the schedule TSN employs a network-wide clock synchronization protocol, namely IEEE 802.1AS, that allows for sub-microsecond precision. Subsequently, the platform provides a TSN Ethernet switch device (the TSNCard component in Fig. 6) that enables the FN to communicate deterministically over Ethernet but also acts as an infrastructure node providing switching capabilities. A wireless TSN card (the WirelessCard component in Fig. 6) allows for deterministic wireless communication. Wireless TSN is envisioned both for WiFi 7, more specifically IEEE 802.11be [71] and 5G [72]. Both of these standardization efforts, IEEE 802.11 and 3GPP, cover in detail industrial use cases for wireless communication.

Finally, the platform provides a fieldbus interface, as a compatibility feature, to enable communication with common industrial devices and actuators. Special consideration is taken in the possible difference of the time domains between the TSN switch and the wireless TSN. The wireless TSN card is responsible for channel access for all wireless devices with a shared medium within a network under strict requirements on reliability and timeliness following different protocols such as contention-based protocols, contention-free protocols, and hybrid protocols [73]. Moreover, this component combined with a TSN card is to exchange data for the hybrid wired–wireless connections. Here, we consider a hybrid protocol TDMA–NOMA (Time Division Multiple Access–Non Orthogonal Multiple Access) that can help to reduce end-to-end latency as well as to increase reliable communication. For example, with such an approach, IIoT sensors transmit the collected data using TDMA, where one node transmits per time slot, and NOMA is used to allow multiple nodes to transmit in the same time–frequency resource block. See [74] for a discussion of advantages, disadvantages and myths about NOMA in this context.

4.2. Resource self-management, orchestration and self-configuration techniques

The FORA FCP reference architecture provides the means for resource self-management, orchestration, and self-configurations. Orchestration helps to align the resource demands of applications and the resource supply of different fog nodes with each other, i.e., to avoid that single applications utilize fog nodes in a greedy way [31]. Instead, resources are composed, i.e., combined with each other. It is the goal of the FORA FCP reference architecture to make sure that the overall system landscape is well-balanced and that all applications are provided with the necessary computational resources (via fog nodes). For this, it is taken into account which requirements the applications have, e.g., if they have real-time demands or not. Based on these demands, the orchestration and resource management capabilities of the fog nodes compute solutions for task scheduling and resource allocation. In general, orchestration and self-configuration techniques need to be provided on the system-level (i.e., for the fog nodes), and also take into account the networking perspective [75]. Importantly, resource allocation and task scheduling are not done on a single level, e.g., separately just at the edge of the network or just in the cloud. Instead, the FORA FCP provides cross-layer resource allocation, so that resources from the edge of the network to the cloud can be exploited if necessary and based on the demands of the applications.

As it is state-of-the-art in fog solutions, resource allocation is not done for fixed settings, but explicitly takes into account the volatility of fog landscapes, where nodes may enter or leave a landscape (or a network) at any point of time, and where the connections between nodes may also change during runtime [76]. For the resource allocation and task scheduling, functional and non-functional requirements are taken into account. The latter includes quality-of-service aspects like latency or security, but also the occurring cost of a particular resource allocation and task scheduling plan.

Resource allocation and task scheduling provide loosely-coupled functionalities, i.e., different methods and algorithms might be integrated into the FORA FCP, and it is even possible to provide transition mechanisms from one method/algorithm to another [77]. In contrast to other fog architectures, the FORA FCP explicitly foresees that fog landscapes may be organized in different ways, with hierarchical vs. fully decentralized landscapes being the two most extreme ways to organize a fog landscape [27].

While hierarchical landscapes are today the state of the art, fully decentralized landscapes more closely mirror the basic architecture of the IIoT. However, since most approaches to resource allocation and task scheduling are based on a hierarchical fog, novel approaches for decentralized landscapes need to be developed. To automate the distribution of applications and exploitation of computational resources in a fog landscape, the FORA FCP foresees that configuration tools are able to set up the single fog nodes based on the outcomes of the resource allocation and task scheduling computations, as well as further requirements.

Furthermore, for many of the functionalities mentioned here, it is necessary to monitor the nodes, in order to know their status. Hence, the FORA FCP provides the means to integrate monitors on different levels, e.g., for the cores or single tasks. Since fog (and IoT) landscapes are inherently volatile, faults may occur at any point in time. In order to be able to mask or mitigate failures in a fog landscape, the FORA FCP allows to allocate applications to new computational resources, even during the runtime of the system. For this, mechanisms which allow to store and re-establish the state of applications are necessary. Last but not least, fog landscapes should be able to be integrated with non-fog (legacy) systems. Especially in Industry 4.0 scenarios, OPC UA plays an important role [78], while the Data Distribution Service (DDS) is an important technology applied in real-time systems [79]. Therefore, the integration of fog nodes with OPC UA and DDS will be needed so that the FORA architecture does not provide a closed system, but is able to integrate other technologies to augment the fog, if meaningful.

4.2.1. AADL models

The proposed fog platform has four major building blocks: (i) the hypervisor that can host OSes, virtual machines or containers, (ii) services for allocation and management of local resources and other essential services (e.g., TSN management), (iii) configuration services for the fog node, and (iv) the orchestration component that enables communication and resource sharing among fog nodes.

As mentioned, the platform is also capable of hosting *containers*, which are a lightweight virtualization alternative, e.g., Docker is a widely-used container technology for operating-system-level virtualization [80]. Containers benefit from the fact that they share kernel functions of their host (i.e., they do not require separated operating systems running in each of the containers unlike virtual machines). This introduces the following advantages: (i) rapid boot time of containers, (ii) higher computational performance, and (iii) lower overhead. Containers offer self-healing mechanisms (i.e., prompt restarting of faulty containers) and mechanisms to increase dependability (containers can

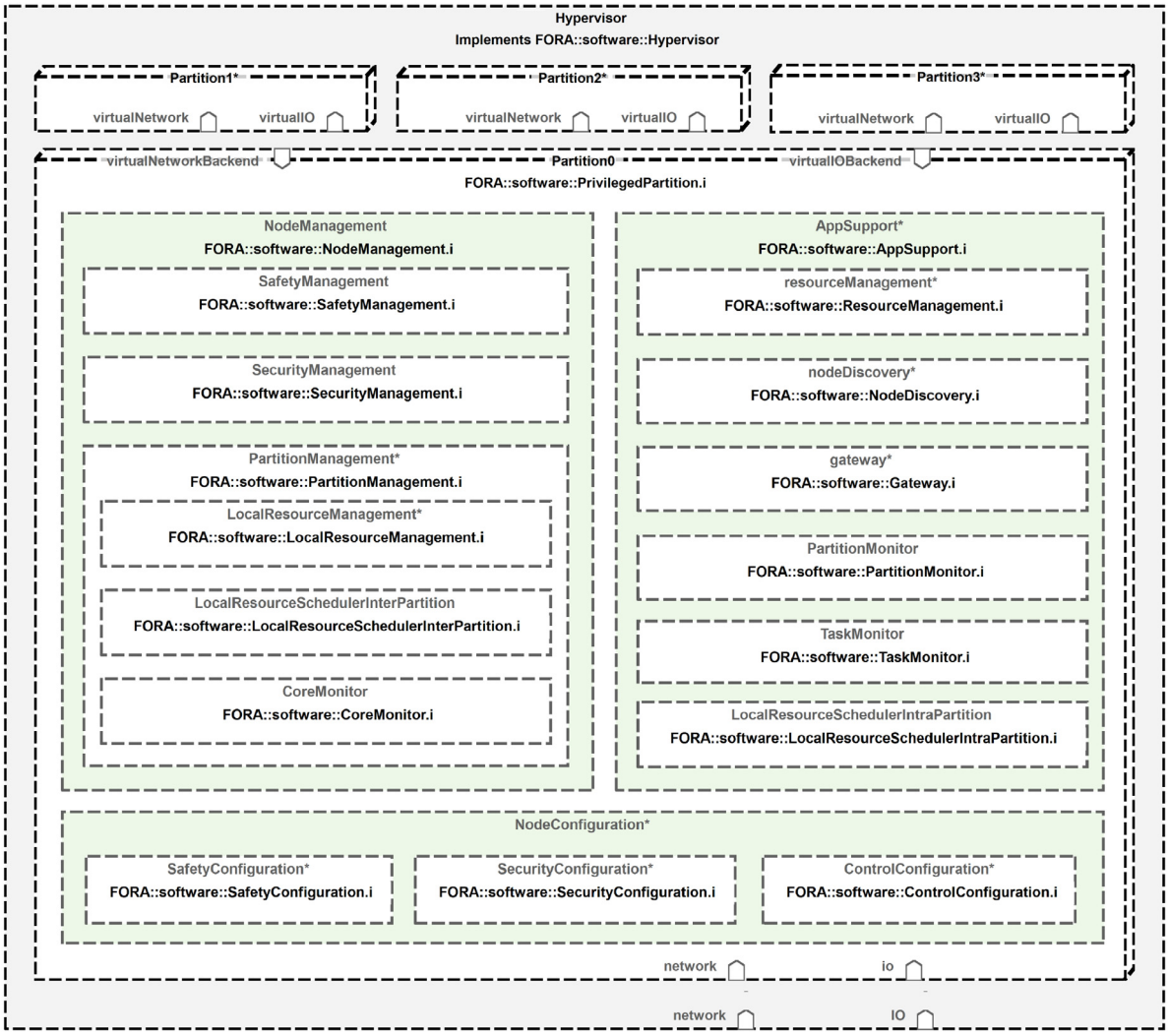


Fig. 7. Overview of FCP's resource management, orchestration and configuration components.

run in multiple instances). The main drawback of container-based virtualization is weaker resource isolation and potential lower level of security [81]. Although containers have typically been deployed on top of a rich OS, our FCP is also able to support them on top of separation kernels such as PikeOS; recent research has also extended containers with real-time capabilities.

In the following paragraphs, we provide a detailed description of the components used in the AADL model from Fig. 7. The Node Configuration component provides the means for configuration which are used by the Node Management component to control the necessary settings for tasks. For example, the Control Configuration component provides the configuration for temporarily separating tasks with different criticality levels including the control tasks that have the highest criticality level, see [82] for more details. Additionally, the Node Management component contains a Security Management component. This component is responsible for the configuration of critical security mechanisms, such as the configuration of key distribution infrastructure. The component employs the system call instrumentation as proposed in [83] to monitor the entire system at runtime, which provides a configuration that determines the security functionality available to applications and services. It enables nodes to communicate with each other and external services securely, by defining protocols that can be used to uniquely identify peers, and to establish secure mutually authenticated communication channels. It also defines a set of acceptable data encryption algorithms for confidential data storage, in accordance with predefined security levels. These then are available as primitives to application services, enforcing node-wide consistent data security. Further, it manages the setup of firewalls, enabling host protection.

The AppSupport component provides the orchestration functionality to the FCP and consists of three different components, i.e., resource management, node discovery, and gateway. Each component contributes to the efficient use of the available resources in the network and the connection with different middlewares. A special focus lies in providing a deployment strategy for mapping the application's tasks to fog nodes, the discovery of the network topology, and establishing the communication between networks.

Resource management aims at deploying new applications on the FCP by creating a collaboration between fog nodes where each has the possibility of making local decisions regarding what tasks of the application to execute, see [84] for more details. This component consists of two distinct modules (i) a resource allocation module and (ii) a network monitoring module. The latter is set as a requirement for the FCP and aims at monitoring the network to provide the communication latency between different nodes. The former makes use of two new modules, i.e., the deployment module and the decision module, being responsible for finding a satisfiable mapping on the FCP for the deployed application.

Besides the internal modules, the resource management component requires extra information about the fog nodes as well as providing access to different middlewares found in the network; information that is provided by collaborating with the remaining AppSupport components, i.e., nodeDiscovery and gateway.

The nodeDiscovery component provides the resourceManagement component with the candidate nodes for deploying applications. To do that, this component integrates node discovery algorithms such as [85], that store a set of neighbor nodes which can be used for distributing an application among various distributed fog nodes. Thus, nodeDiscovery is an essential part of the AppSupport component because it discovers all the fog nodes of the system, and makes their resources available to the resourceManagement component which handles the deployment of the applications. Upon the discovery of new nodes, this component selectively chooses which nodes become neighbors. This is done based on proximity measurements (e.g., using round-trip time or hop count [85]) in order to enable the execution of applications with low communication delay and therefore overall execution time. While currently based on a best-effort approach, the component could also implicitly take into account execution times of applications based on user requirements. Furthermore, the selection of neighbors determines the overall communication among the nodes, which can be either hierarchical or peer-to-peer. The hierarchical communication type enables the nodes to communicate in a tree-like topology which is based on layers. In the peer-to-peer communication type, the nodes communicate based on a flat model which does not use layers [86]. The selection of the communication type is based on the requirements of the applications.

The gateway enables a fog node to communicate beyond its own network by connecting OPC UA and the DDS. After the resourceManagement component deploys the application in a fog node, the user needs to configure the gateway by using a configuration file. The gateway configuration is static and is dependent on the configuration of the OPC UA and respectively DDS configuration. The OMG gateway specification document [87] provides further details about the gateway and its internal functioning.

The CoreMonitor component monitors the core status. This component is a piece of software in charge of informing the LocalResourceManagement that a core is still alive. It is assumed that the core fail mode is not running. The implementation can be done in different ways, but in all of them, it must run from the privileged partition, as it must be aware of which partition is running and in which core. The core status information is essential to take local and global decisions in case of failure.

The PartitionMonitor component monitors the partition status. This component has a similar function to the CoreMonitor, but with the objective of monitoring the partitions. We assume here that the partition fail mode is a total failure, that means, the partition does not run anymore. Since this component must be aware of each existing partition and its status, it also must run from the privileged partition. This component can be part of the CoreMonitor if the according monitoring objects are strictly connected, however, the information sent to the LocalResourceManagement has to be different. The TaskMonitor component monitors the critical task execution status and progress. Although the hypervisor can guarantee temporal and spatial task and partition isolation, in a COTS MCP, without previous detailed knowledge of all the tasks that can run at the same time in different cores, it is not possible to forecast how the inter-core interference delays the execution of the tasks due to the physical memory sharing among the cores. To this end, similar to the proposed approaches such as [88], we implement the PartitionMonitor component for managing the shared resources on the platform. To guarantee critical task deadlines in this kind of hardware platform this component has to be implemented in such a way where it provides information to the LocalResourceManagement and LocalResourceSchedulerInter-Partition so that it can suspend non-critical partitions to avoid missing critical task deadlines.

The LocalResourceManagement component is in charge of gathering the information from the components described above, namely: CoreMonitor, PartitionMonitor, and TaskMonitor. This component is also in charge of communicating the status of the resources to the following components: (i) LocalResource SchedulerIntra-Partition, (ii) LocalResourceSchedulerInter-Partition, and (iii) ControlConfiguration. This component must also run from the privileged partition since it has to have access to the other components in the entire platform.

Considering that the critical tasks and partitions are scheduled by the ControlConfiguration component, the LocalResource-SchedulerInter-Partition is in charge of finding feasible non-critical partitions scheduling according to the workload and locally available partitions. Taking into account the dynamic behavior of the task requests in a typical industrial automation scenario, this component is also in charge of controlling the non-critical partitions at runtime, suspending, running or changing the corresponding CPU occupation time. To schedule the partitions, this component has a strong interaction with the LocalResource SchedulerIntra-Partition.

Working tied to the LocalResourceSchedulerInter-Partition, the LocalResourceSchedulerIntra-Partition component is responsible for finding a feasible non-critical task schedule according to the workload. As the previous component, to meet the dynamic requirements of a real industrial automation scenario, it is able to find a new schedule at runtime. This component is independent of the scheduling policy and it is possible to have different policies applied for different partitions.

4.3. Applications and services

The FCP has to be both agile and dependable. IT and OT worlds have different focus on dependability attributes such as safety and security, which have to be reconsidered in the converged IT/OT FCP. Regarding safety, when a system has the potential to harm humans or the environment (or is intended to mitigate or manage such harm), decision-makers require safety assurance evidence that it manages the risks acceptably. The conceptual basis for certification is that the evidence anticipates the possible circumstances that can arise from the interactions between the system and the environment, to show that these interactions do not pose an unacceptable risk. Hence, the FORA FCP integrates approaches, developed as platform services, for assuring the safety and security of the FCP. In addition, the FN's proximity to the sensors and machines is an opportunity for improved data analytics, which, together with monitoring, improve agility when they support fast decision-making and resource allocation at the edge.

Security and privacy in OT lags behind IT, as the current state of practice is to use "air gaps", physically isolating sensitive equipment (locked doors and guards) from unsecured networks, which, with the introduction of standard IT systems no longer works. For IT, *security* and *privacy* are important, together with reliability, but safety is not considered. The convergence of IT and OT brings new security challenges, exposing previously isolated OT to new types of attacks [89]. Fog Computing introduces new security and privacy challenges that need to be addressed in order to promote such a new computing paradigm. Fog Computing inherits the Cloud Computing security issues, but these are more critical due to the safety issues of industrial systems. With the exception of very few preliminary papers, research is still immature [90].

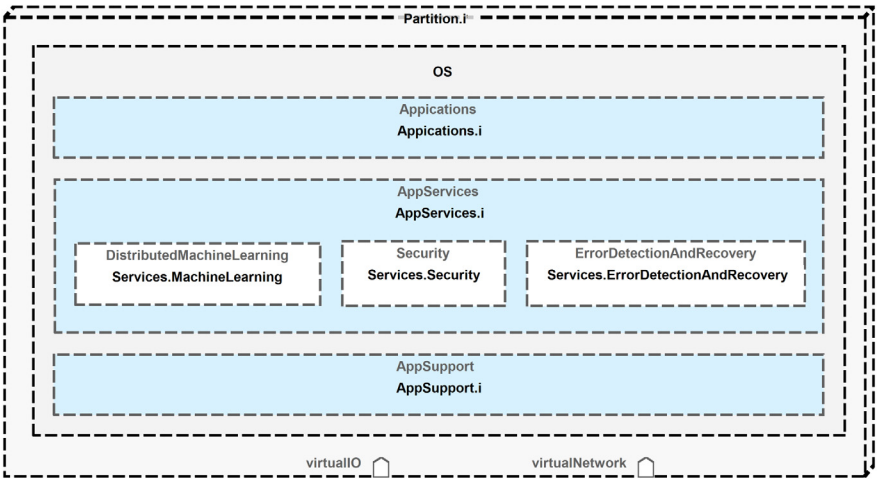


Fig. 8. Overview the FCP's services.

Big Data and *data analytics* drive novel applications in Industry 4.0 [91]. Sensors and machines generate huge amounts of data: industrial data is growing faster than any other sector and manufacturing stores more data than any other sector. Access to industrial data is difficult because of the different data representations used by sensors and machines, data variety, and data velocity, e.g., the speed at which data is generated. Many applications enable data analytics by storing historical data in the Cloud, and running Machine Learning (ML) algorithms to gain insights that lead to value creation. Existing ML solutions from, e.g., IBM, Google, Microsoft, do not address industrial automation because large amounts of data cannot be moved to the Cloud due to bandwidth constraints, and with OT there are severe computation and storage limitations as we get closer to the machines. There is limited work in applying ML in real-time and on distributed data streams, e.g., distributed learning [92].

Regarding application development, the Cloud Computing programming model typically follows a Service-Oriented Architecture (SOA) model. Cloud applications are written using interacting (micro)services, developed as containers which rely on scalable Platform-as-a-Service (PaaS) [93]. Although there are many PaaS solutions for the Cloud, no such solutions exist for the Fog, where applications may reside in a continuum involving the machines, in the Fog Nodes, and the Cloud. In addition, critical applications with real-time properties cannot be modeled.

To address these challenges, our AADL platform model includes the following services, depicted in Fig. 8: resource monitoring services with extended insights into the individual nodes and network, and safety and security monitoring for recovering from failures and attacks; a new framework for edge analytics that goes beyond the traditional cloud-based post-collection analytics model that decouples data acquisition from knowledge extraction, improving thus the resource-efficiency of the FCP and IIoT end-devices, enabling real-time decision making, intelligent filtering, and resource prioritization; services for securing the FCP, based on the concept of Security-by-Contract and compliant with IETF MUD (Manufacturer Usage Description) [94], in order to be applied to the vast majority of IIoT devices, addressing the heterogeneity challenge.

4.3.1. AADL models

The AADL components related to the discussion in the previous section are mainly of two categories, (1) Components that are essential for assuring the required levels of dependability attributes such as timeliness, safety and security and typically implemented in the platform, and (2) components that specifically target the application services and enable a coherent methodology for implementation of them. Category-1 involves some of the components that have already been explained as part of Sections 4.1 and 4.2. We will contribute in designing as well as extending their scope w.r.t. enabling fault tolerance, timeliness, security and safety aspects. In this subsection, we focus on the components that are of Category-2, i.e., implementing application-level services in Fig. 8.

We are developing appropriate intrusion detection techniques to secure the execution of mixed-criticality applications on the fog node. The intrusion detection module collects system events, such as hardware performance counters and system calls [83]. It analyzes them with Machine Learning techniques to detect anomalous patterns of execution. The solution is non-intrusive, since it observes the monitored software's interface without direct interaction and hence will not affect the predictability in any adverse manner. The intrusion detection performance overhead is limited due to the embedded environment properties. Because it is deployed locally on the target, its execution should not interfere with other system applications, which potentially have safety requirements and deadlines. It also must not be reachable by other untrusted entities to preserve its own security. Therefore, the secure integration of intrusion detection module in a partitioning based system design for embedded mixed criticality environments is also essential. These techniques are implemented partly in the hypervisor to have direct hardware access (see Fig. 7) and partly in the Security component in Fig. 8.

Due to the increasing focus on TSN in the provision of predictable network traffic, we are developing a fault detection, isolation, and recovery (FDIR) method to be applied in the context of TSN communication and task's executions, similar to [95]. The method will reside in the middleware and will monitor tasks executions and network communication traffic. It applies fault detection and identification techniques, and enables appropriate recovery mechanisms in case of faults. IEEE 802.1CB TSN standard provides fault-tolerance by means of stream redundancy i.e., splitting streams across disjunctive links to maximize probability of correct reception in the presence of link faults. We are also exploring the feasibility of a Fault Tolerant Communication Configurator, which is capable of configuring the network stream transmissions in an optimal manner to ensure that resources are conserved while at the same time, fault-tolerant guarantees are provided. Similar to the security techniques, these fault-tolerance techniques are implemented both at lower levels (within the hypervisor and RTOSes) and in the ErrorDetectionAndRecovery service depicted in Fig. 8.

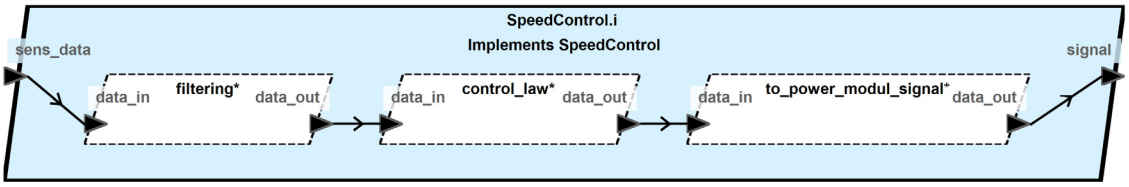


Fig. 9. An example of modeling a critical application with three critical tasks using FORA AADL models.

A fog platform brings the computing power from the remote cloud-side closer to the edge devices to reduce latency, as the unprecedented generation of data causes ineligible latency to process the data in a centralized fashion in the Cloud. In this new setting, edge devices with distributed computing capability, such as sensors and surveillance cameras can communicate with fog nodes with less latency. Furthermore, local computing (at edge side) may improve privacy and trust. Hence, we integrate a new method to decompose the data processing, by dividing them between edge devices and fog nodes, intelligently. We apply active learning on edge devices (see [96] for more details); and federated learning on the fog node which significantly reduces the data samples to train the model as well as the communication cost, similar to [97]. However, this work has been used to implement a Predictive Maintenance framework, which uses distributed machine learning, where the distributed drivers and the centralized server jointly (collaboratively) train one global model. An Example of using machine learning methods for Predictive Maintenance can be found in [98]. Typically, the decentralized drivers placed in different locations, generating data that captures the local information instead of global information and they train their local models based on partial knowledge. The aggregation step at the server-side enables the information sharing between drivers and server to obtain one model with overall knowledge. Then, the server sends the aggregated model back to drivers. The edge analytics service is implemented in the DistributedMachineLearning component from Fig. 8.

Regarding mixed-criticality application modeling, critical and non-critical applications can be defined in FORA FCP architecture as AADL process and thread components. Non-critical applications can be defined as standard process and thread components in AADL, whereas critical applications should be defined as extensions of CriticalApp and CriticalTask components introduced by FORA AADL models, which are extensions of process and thread components. Critical applications may be control applications with quality-of-control requirements, safety-critical applications with dependability requirements (e.g., have to be replicated for fault-tolerance reasons) or real-time applications with soft or hard real-time properties. All of these applications are real-time and are modeled using the typical sporadic task model from real-time theory [99], where each periodic task has a period (minimum inter-arrival time for sporadic tasks), a worst-case execution time and a deadline. We opted to use the timing properties proposed by SAE in the Timing_Properties property set, which have been developed for capturing such timing requirements. Fig. 9 shows an example of modeling a critical application with three critical tasks. The AADL source code for this application with the detailed specifications for one of its tasks is presentedin Listing 1 to show how the timing properties of an application and a task can be specified.

```
process SpeedControl extends FORA::core::
    CriticalApp
features
 sens_data : in data port sens_data {arinc653::
    sampling_refresh_period => 10 ms;};
 signal : out data port signal {arinc653::
    sampling_refresh_period => 10 ms;};
properties
    FORA::core::CriticalApp::SIL => 3
end SpeedControl;

process implementation SpeedControl.i
subcomponents
 filtering : thread filtering;
 ...
connections
 c0 : port sens_data -> filtering.data_in;
 c1 : port filtering.data_out -> control_law.
    data_in;
    ...
end SpeedControl.i;

thread filtering extends FORA::core::CriticalTask
features
 data_in : in data port sens_data;
 data_out : out data port filter_data;
properties
   dispatch_protocol => periodic;
   period            => 10ms;
   deadline          => 10ms;
   compute_execution_time => 100ms..200ms;
end filtering;
```

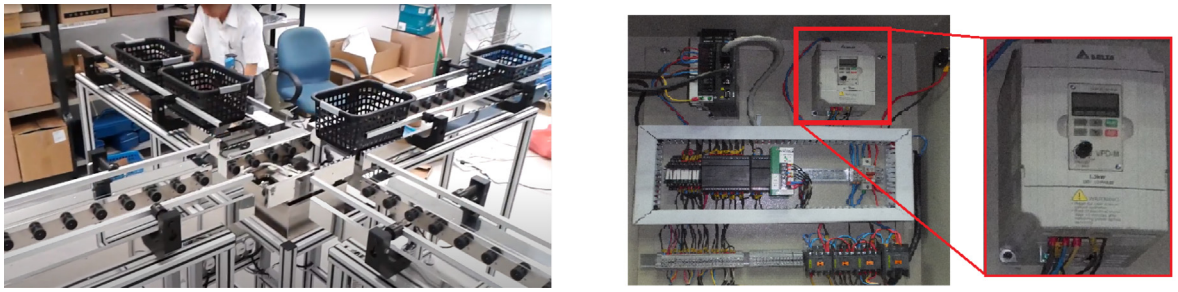
Listing 1: Part of an AADL source code to specify a critical application SpeedControl from Fig. 9

5. Evaluation

Section 3 has outlined the methodology used for the definition and evaluation of the proposed Fog Computing Platform reference architecture. In this section we report the results obtained when modeling, implementing and evaluating a *Conveyor Distribution System* Use Case (UC1 mentioned in Section 3). The details of UC1 have been presented in [12]. Here, we extend that work to: show how the AADL from Section 4 can be used to model the UC; show how the FORA Technology Bricks can be plugged into this AADL model to implement a prototype; evaluate the ability of successfully implementing the UC using the proposed reference architecture. Use Cases UC2 and UC3 mentioned in Section 3 have also been used to evaluate the proposed architecture, as reported in [13,14], but without a focus on AADL modeling.

In UC1, a conveyor distribution system is used to distribute packages from an inventory to different destinations. The conveyor distribution machine is well-known and widely used in inventories for the automatic distribution of packages. Let us consider a typical machine, as depicted in Fig. 10a. The machine is fed with packages from one side and reads the tag of the received package. It gets the destination of the package by accessing a database with the read tag and drives the package towards the destination from one of the other sides of the machine.

Conveyor distribution uses electric motors and drives. *Electric drives* alter the frequency and voltage of an electric motor's current for different rotation speed, torque, and position of its shaft using the implemented real-time software controlling the power electronic circuits [100]. An electric drive in an industrial setting is shown in Fig. 10b. Industrial controllers (e.g., PLCs) sitting on the "Control level" of the automation pyramid determine the required output of electric motors which sit on the "Machine level" (see Fig. 1). The electric drives placed close to electric



(a) Conveyor belt image (b) An electric drive (in red) shown in an industrial setting

Fig. 10. Overview of Conveyor belt-based distribution system demonstrator for UC1.

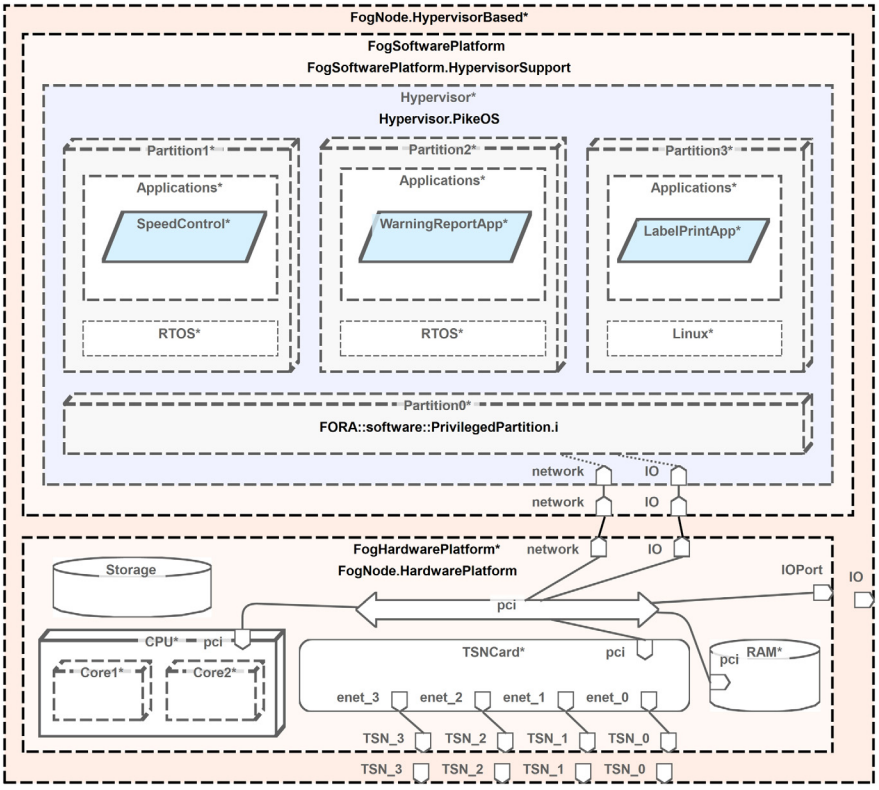


Fig. 11. Configuration of a fog node in the UC's architecture.

motors on the "machine level" convert the determined output to a corresponding electric current to drive the electric motors. Thus electric drives produce massive and critical data about the controlling machinery. Sending all the data to other computing nodes over the Internet or the control network would consume bandwidth and it is also discouraged by the factory owners for confidentiality concerns. Hence, in this use case, we extend the electric drives to serve as fog nodes, leading to new offerings like programmability, local analytics, and connectivity to customer Clouds.

5.1. UC model

We model the UC's architecture with AADL by using the FORA AADL components from Section 4 and refining them to reflect the UC requirements. We model positioning and tag-scanner sensors as *Sensor.Position* and *Sensor.tag*, respectively; and the electric motors as *Actuator.PowerModule* and *Actuator.ElectricMotor*. We model the TSN switches as *Swith.TSN_SW* and configure them using FORA AADL property set developed for configuring a TSN network, which allows specifying routing specifications and message schedules for a switch as system properties. We model the fog nodes as *FogNode.FN* and configure them by refining the software and hardware subcomponents of the *FogNode* model. Thus, Fig. 11 presents the AADL model of a Fog Node. The FN consists of a hardware platform which has a multicore processor, a TSN enabled network switch and I/O to an external power module for generating the electric current and a software stack which has a hypervisor, a middleware, and partitions with dedicated operating systems (real-time OS for running control applications and Linux/Windows-based OS for running best effort applications) and application layers, respectively. The fog nodes are assumed to run mixed-criticality applications including control applications for controlling electric motors. The mixed-criticality applications are temporally isolated using partitions which are managed by the hypervisor i.e., PikeOS in this UC which supports static partition tables (see [82] for more information about partition tables and enforced isolation). Finally, we model the UC platform as a system that consists of the required instances of the aforementioned components connected via TSN. Fig. 12 presents

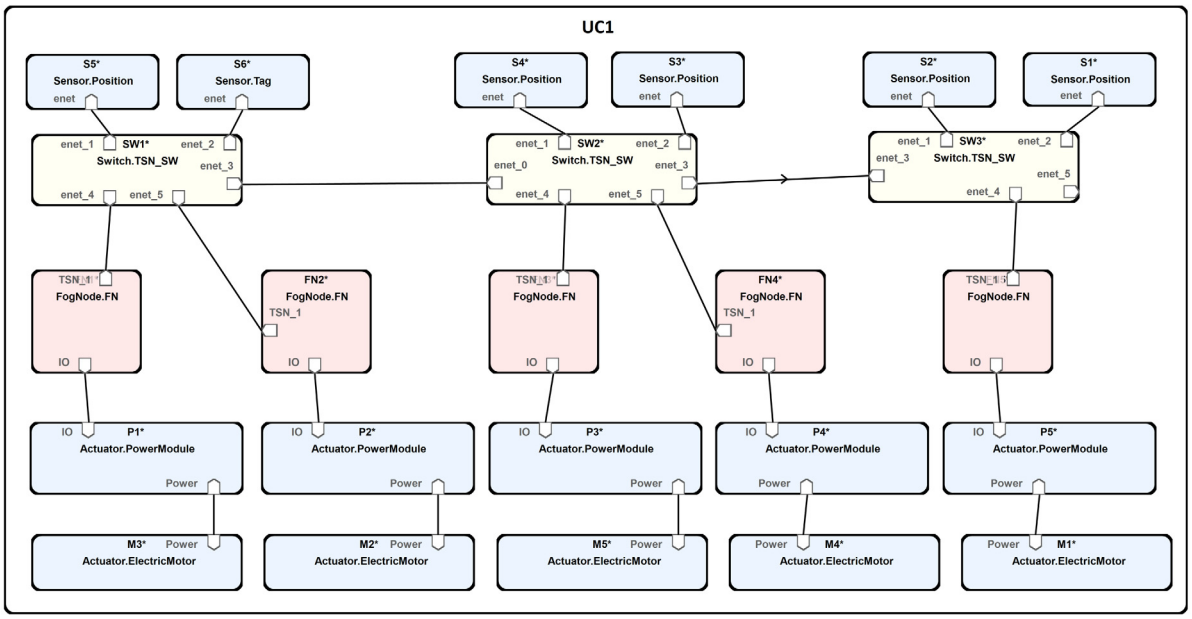


Fig. 12. AADL diagram of the UC's architecture.

the resulted AADL architecture model for the UC. As it can be seen, the architecture consists of three switches (SW_i, denoted in the following text with the notation $\mathcal{W}$), five fog nodes (FN_i, denoted with $\mathcal{E}$), five position sensors and one tag scanner sensor (all denoted with S). Each fog node is connected to the electric motor (m_i) it controls.

We model the applications using AADL as discussed in Section 4.3.1. For space reasons, instead of showing the full AADL model, we instead use a table where we present the properties of the applications, see Table 2. In this UC each fog node $\mathcal{E}_i$ runs a control application which receives a message from its corresponding sensor S_i and controls the belt to drive the package by determining the required corresponding electric motor's output. The fog node also runs an electric drive application which sets the electric motors current for the external power module via I/O. Thus, the UC has five control applications (one for each fog node), and each control application gets one stream as input and sends its output 10 ms after receiving the input (we assume that the outputs have fixed offset from the inputs and have no latency).

The details of the applications are shown in Table 2 where the column 1 shows names of the applications. The applications 1 to 5 are control applications for controlling the speed of electric motors, and hence have control performance requirements. Each application has a criticality Level (L)—that can represent the Safety Integrity Levels (SIL) of the application, with values from 0, non-critical, to 4, highest criticality [101] (see the column 2 in Table 2), a number of tasks (shown in the column 3) with the same period (P) (shown in the column 4), and a computation cost (C) which is the sum of worst-case execution times of the tasks over their period (shown in the column 5). We assume that all the applications (including the control applications) interact via a set of streams which have hard real-time requirements, and all the streams are prioritized concerning their criticality. The details of the streams (size S column 7), period T column 8, and routing column 9 are shown in Table 2. We assume that all links have the data rate of 100 Mbps. Listing 1 presents the AADL specifications for one of these applications (speed control).

5.2. Implementation model of network traffic for QoC

The UC's architecture is configured to guarantee the timing requirements of all the network streams on TSN, including the control applications' streams which have more stringent timing requirements. The configuration is composed of Gate Control Lists (GCLs) for the TSN network switches which represents the message schedules, and is provided by the ControlConfiguration component (see Section 4.2.1). The GCLs regulate the network traffic such that functional timing requirements of network streams i.e., stream deadlines, and their non-functional timing requirements are met. The control performance requirements of control applications is defined as Quality-of-Control (QoC) (see [61] for more details). We evaluate these requirements by analyzing the GCLs.

We employ a Constraint Programming-based schedule synthesis strategy, implemented as an OSATE plug-in, aiming at maximizing the QoC and satisfying the deadlines of network streams, proposed in [61] to generate the GCLs. Thus, all the streams have been successfully scheduled, i.e., none of the deadlines is missed. The configuration has also provided the minimum delay and jitter for the streams, resulting in a good control performance. The column 10 (last but one) in Table 2 shows the maximum end-to-end delay (ED) of streams. We used JitterTime [102] to simulate the behavior of the control applications which reports an average value of 0.009 for the QoC of all control applications, i.e., a good control performance (see [61] for the exact cost function).

5.3. Implementation model of hypervisor partitions and task schedules for QoC

Mixed-criticality applications sharing the same platform have to be isolated from each other, otherwise a faulty lower-criticality task may interfere with a higher-criticality task, leading to failure. In the UC's architecture, each fog node uses a deterministic hypervisor for virtualizing the applications by providing deterministic access to shared resources via a static configuration table. The deterministic access realizes the temporal isolation for the applications with different levels of criticality via partitioning aiming at protecting the applications from the possible interference.

The ControlConfiguration component (see Section 4.2.1) uses the heuristic algorithm proposed in [61] to allocate a partition for each criticality level of the assigned applications, and to decide

Table 2
UC1: applications, streams and evaluation results.

Application	L	Tasks	P (ms)	C (μs)	Relevant stream	S (bytes)	T (ms)	Routing	ED (μs)	ED after TESLA (μs)
m_1 control	3	3	10	0.35	S_1 data	700	10	$S_1 \rightarrow \mathcal{W}_1 \rightarrow \mathcal{E}_1$	60	1241
m_2 control	3	3	10	0.35	S_2 data	850	10	$S_2 \rightarrow \mathcal{W}_1 \rightarrow \mathcal{E}_2$	72	1481
m_3 control	3	3	10	0.35	S_3 data	600	10	$S_3 \rightarrow \mathcal{W}_2 \rightarrow \mathcal{E}_3$	52	1111
m_4 control	3	3	10	0.35	S_4 data	950	10	$S_4 \rightarrow \mathcal{W}_2 \rightarrow \mathcal{E}_4$	80	2407
m_5 control	3	3	10	0.35	S_5 data	500	10	$S_5 \rightarrow \mathcal{W}_3 \rightarrow \mathcal{E}_5$	44	921
Package status	2	2	10	0.3	m_2 state	1100	20	$\mathcal{E}_2 \rightarrow \mathcal{W}_1 \rightarrow \mathcal{E}_1$	152	1911
Motor break	3	3	10	0.35	–	–	–	–	–	–
Table break	2	1	8	0.31	–	–	–	–	–	–
SCADA	1	4	10	0.28	$\mathcal{E}_5$ data	920	10	$\mathcal{E}_5 \rightarrow \mathcal{W}_3 \rightarrow \mathcal{W}_2 \rightarrow \mathcal{E}_4$	254	2389
User interface	1	3	6	0.28	–	–	–	–	–	–
Database access	1	8	15	0.59	S_6 data	1200	30	$S_6 \rightarrow \mathcal{W}_3 \rightarrow \mathcal{W}_2 \rightarrow \mathcal{E}_3$	200	3091
Weight report	2	5	15	0.47	–	–	–	–	–	–
Warning	2	2	10	0.26	$\mathcal{E}_4$ data	700	50	$\mathcal{E}_4 \rightarrow \mathcal{W}_2 \rightarrow \mathcal{E}_3$	260	1751
Destination set	1	3	8	0.56	m_2 set	850	50	$\mathcal{E}_3 \rightarrow \mathcal{W}_2 \rightarrow \mathcal{W}_1 \rightarrow \mathcal{E}_2$	144	2221
Label print	1	4	12	0.59	–	–	–	–	–	–

mapping of the partitions to cores of the fog nodes. The component generates schedule tables for partitions and for tasks inside each partition considering the determined mapping of tasks to the partitions and cores. A schedule table s_i repeats periodically with a system cycle and captures the start and finishing time of tasks (or partitions for partition tables). The generated task schedules are optimized for QoC of control applications which are assumed to have high-criticality levels.

We evaluate the performance of the configuration in providing temporal separation and preserving QoC for control applications. We assume that each fog node has a dual-core processor. The ControlConfiguration component has determined the mapping of partitions to the cores of fog nodes. The component has also successfully scheduled the partitions and all the tasks inside the allocated partitions. The results show that none of the tasks has missed its deadline and all of the tasks are isolated concerning their criticality levels by mapping them to the partitions with same criticality levels. The results show that the cores have average utilization value of 57.2% and the maximum utilization value is reported as 73.6% for the core 0 of the fog node $\mathcal{E}_3$. Furthermore, the control application has a good control performance, which is evaluated with JitterTime [102] that calculates a value of 0.4103 for the QoC (cf. the cost function from [82]).

5.4. Addressing extensibility for dynamic fog applications

The UC consists of statically allocated critical applications to run on the platform. These applications are statically mapped to the cores and partitions, and scheduled inside the mapped partitions at design time. The UC's architecture is also capable of running dynamic non-critical applications which can migrate in-and-out of the fog nodes over time and be removed/replaced by other such applications. This capability, known as extensibility, is realized in a way that the design time configuration is not modified which is a necessity for keeping the performance level of the statically allocated critical applications as well as avoiding the safety re-certification of critical applications [103].

The extended configuration is provided by the NodeConfiguration component (see Section 4.2.1) at runtime and schedules the dynamic non-critical applications on their arrival. To allow more dynamic non-critical applications to be added at runtime without negatively impacting the performance of existing applications, the design time configuration needs to be optimized for extensibility which is realized by distributing the idle time of the design time schedules. The idle time spaces of these schedules are used to accommodate tasks of dynamic applications. Less-deviated idle time duration enables the fog nodes to accommodate more dynamic non-critical applications.

The design-time extensible schedules are generated using the method proposed in [103] and implemented as an OSATE plug-in. We evaluate the extensibility of the UC's schedules by optimizing the generated schedules in Section 5.3. We use the algorithm introduced in Section 5.3 to schedule the tasks from Table 2. The algorithm generates schedule tables for the cores of the fog nodes in the UC. We take the schedule s_8 (representing the schedule table on the second core of fog node $\mathcal{E}_4$) and depict it in Fig. 13a (as "BASE") using a Gantt chart, where the boxes are execution slices, and the arrows show task preemption. The execution slices are denoted with the task's number. All the tasks have the same criticality level and are scheduled inside the same partition. While "BASE" is not optimized for extensibility, we give an example for optimized version of the same schedule described in Fig. 13c as "OPTIMIZED". We use the same extensibility metric as presented in [103], which reports the values for deviation of idle time duration as 0.0014 and 0.0003 for "BASE" and "OPTIMIZED" respectively, showing that "OPTIMIZED" has less-deviated idle time duration, i.e., it is more extensible.

We consider a scenario where engineers want to add four dynamic non-critical applications to the fog node $\mathcal{E}_4$. These applications represent logging applications. Each dynamic application composed of a single task which has a deadline constraint equal to its period. The applications' periods are 6, 8, 10, 12 ms and their computation costs are 17%, 13%, 15%, 13%, respectively.

Thus, we extend the schedules "BASE" and "OPTIMIZED" by adding the dynamic applications 1–4 to the schedules. The resulting schedules are depicted in Figs. 13b and 13d as "BASE-EXT" and "OPTIMIZED-EXT" respectively. The results show that the dynamic application has successfully scheduled in "OPTIMIZED-EXT" i.e., none of the applications has missed its deadline, whereas some deadlines are missed in "BASE-EXT", for example, the task of the application 1 has missed its deadline at 12,900 ms. The configuration provided by Node Management component shows promising results in successful adding of dynamic applications without modifying the existing schedules, and is able to bring extensibility to the schedules of fog nodes.

5.5. Addressing security requirements in TSN

The UC's architecture requires secure communication for confidential messages to guarantee confidentiality, integrity of the data, and authenticity of the remote party. In the UC, the platform blocks any attempt at communication to endpoints it cannot authenticate.

Since TSN does not provide security mechanisms, we employ the Timed Efficient Stream Loss-Tolerant Authentication (TESLA) [104] to guarantee the security requirements in our proposed architecture. TESLA is a low-resource multicast authentication protocol which relies on synchronized schedules of tasks and messages which is implemented in the SecurityConfiguration

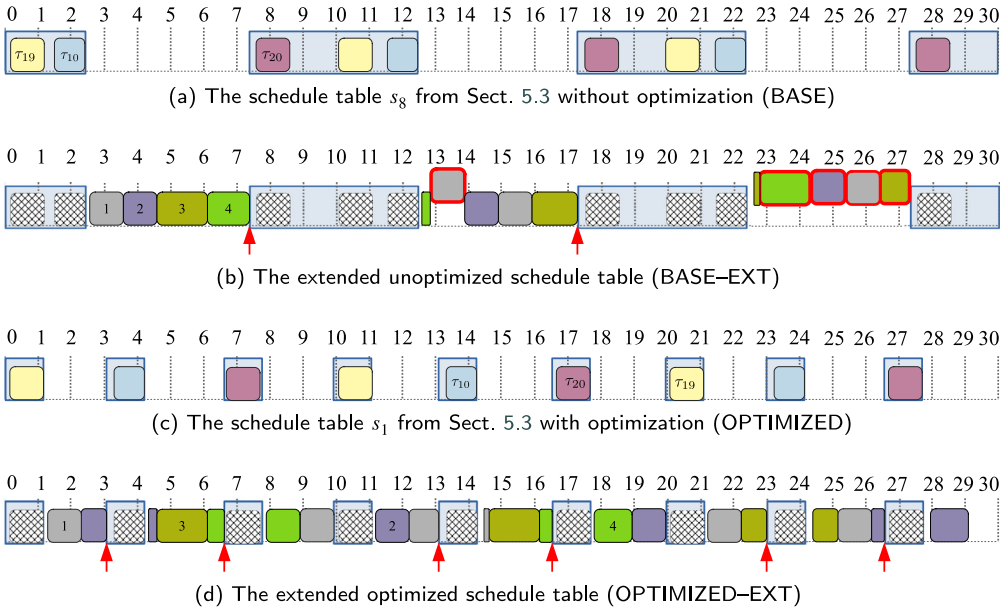


Fig. 13. Four schedule tables (variants of s_8 generated in Section 5.3) for extensibility example: The colored boxes are execution slices; the transparent boxes show the partitions; the red bordered boxes show missed-deadline tasks. The arrows show preemption and the hatched boxes shows occupied time slots. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

component of the AADL model. The authentication protocol uses MD5-MACs of 16 Byte Length and 16 Byte keys. The configuration provided by the component consists of a dedicated security application composed of two tasks, for each network message which is scheduled to run either at the message's reception or transmission depending on the end system that the application is running on.

We scheduled the UC's streams from Section 5.2 using the TESLA method to evaluate the security mechanism of the UC's architecture. The results are shown in Table 2. The Columns 10 and 11 (last two) exhibit the maximum end-to-end delay (ED) of streams before and after adding the security mechanism respectively. The security configuration provided confidential authenticated communication by using MD5-MACs and authentication keys at the expense of increasing the end-to-end delay of messages by 1723 μs on average.

6. Conclusions and discussion

This paper has presented a Fog Computing Platform (FCP) reference architecture aimed at Industrial IoT applications. The architecture was defined and evaluated within an overall methodology that was driven by requirements collected via three IIoT use cases. The definition of the FCP reference architecture has been done via AADL models. We have presented an overview of the models and the entities of the FCP within three main themes: (i) computing device and communication, (ii) resource management and orchestration and (iii) application and services. We have discussed the reasoning and analysis behind the definition of the FCP and listed the major components and the technology bricks developed to implement a design.

The proposed reference architecture was evaluated on a conveyor belt distribution system demonstrator, showing the capability to successfully model and implement IIoT applications. As future work, the FORA AADL models will be further refined and aligned to standards (e.g., IEEE 1934 OpenFog).

A Fog Computing Platform brings several benefits to Industrial IoT applications: End-users benefit from machine interoperability and resource elasticity. The FCP scales on demand to meet business needs and connects all assets of end-users to enable data capture. End-users will be able to connect the machines, the Fog Nodes and the Cloud, allowing optimal resource allocation, driving costs down and value up. Dependable middleware and interoperability protocols make data available for innovative applications, e.g., data-driven decision-making, data analytics.

The FORA FCP provides services needed to rapidly develop, securely deploy and efficiently operate industrial applications. The software platform provides standardization across multiple silos and enables businesses to quickly take advantage of operational and business innovations.

The FCP infrastructure meets stringent industrial regulatory requirements (safety and security), which cannot be met by public Clouds. This reduces the security risks with networked machines. The new virtualized FCP handles security incidents such that the critical operations are not impacted, reducing downtime. New security services help end-users deploy secure industrial applications and detect abnormal or suspicious behavior, recovering from attacks and reducing losses. Data analytics services offer insights, enabling decision-making to increase asset utilization, deploy servicing and maintenance resources efficiently to lower repair costs, plan performance improvements.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] H. Lasi, P. Fettke, H.G. Kemper, T. Feld, M. Hoffmann, Industry 4.0, Bus. Inf. Syst. Eng. 6 (2014) 239–242.
[2] T.J. Williams, The purdue enterprise reference architecture, Comput. Ind. 24 (1994) 141–158.
[3] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.
[4] OpenFog Consortium, OpenFog Reference Architecture for Fog Computing, Architecture Working Group, 2017, pp. 1–162.

[5] F. Bonomi, R.A. Milito, P. Natarajan, J. Zhu, Fog computing: A platform for Internet of Things and analytics, in: N. Bessis, C. Dobre (Eds.), Big Data and Internet of Things: A Roadmap for Smart Environments, in: Studies in Computational Intelligence, vol. 546, Springer, 2014, pp. 169–186, http://dx.doi.org/10.1007/978-3-319-05029-4_7.

[6] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, O. Rana, Fog computing for the internet of things: A survey, ACM Trans. Internet Technol. 19 (2019) 1–41.

[7] P. Hu, S. Dhelim, H. Ning, T. Qiu, Survey on fog computing: architecture, key technologies, applications and open issues, J. Netw. Comput. Appl. 98 (2017) 27–42.

[8] TSN Task Group, Time-sensitive networking, 2017, (accessed July 1, 2020). URL: http://www.ieee802.org/1/pages/tsn.html.

[9] E. Dahlman, G. Mildh, S. Parkvall, J. Peisa, J. Sachs, Y. Selén, J. Sköld, 5g wireless access: requirements and realization, IEEE Commun. Mag. 52 (2014) 42–47.

[10] W. Mahnke, S.H. Leitner, M. Damm, OPC Unified Architecture, Springer Science & Business Media, 2009.

[11] P.H. Feiler, D.P. Gluch, J.J. Hudak, The Architecture Analysis & Design Language (AADL): An Introduction, Technical Report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[12] M. Barzegaran, N. Desai, J. Qian, K. Tange, B. Zarrin, P. Pop, J. Kusela, Fogification of electric drives: An industrial use case, in: Proc. of IEEE International Conference on Emerging Technologies and Factory Automation, 2020a, pp. 77–84.

[13] P. Denzler, J. Ruh, M. Kadar, C. Avasalcai, W. Kastner, Towards consolidating industrial use cases on a common fog computing platform, in: Proc. of IEEE International Conference on Emerging Technologies and Factory Automation, 2020, pp. 172–179.

[14] M.S. Shaik, V. Struhár, Z. Bakhshi, V.L. Dao, N. Desai, A.V. Papadopoulos, T. Nolte, V. Karagiannis, S. Schulte, A. Venito, G. Fohler, Enabling fog-based industrial robotics systems, in: Proc. of IEEE International Conference on Emerging Technologies and Factory Automation, 2020, pp. 61–68.

[15] A.V. Dastjerdi, H. Gupta, R.N. Calheiros, S.K. Ghosh, R. Buyya, Fog Computing: Principles, Architectures, and Applications; Internet of Things, Elsevier, Amsterdam, The Netherlands, 2016.

[16] IEEE, 1934-2018 – IEEE standard for adoption of OpenFog reference architecture for fog computing, 2018.

[17] F. Giust, X. Costa-Perez, A. Reznik, Multi-access edge computing: An overview of ETSI MEC ISG, IEEE 5G Tech. Focus 1 (2017).

[18] C. Puliafito, E. Mingozzi, C. Vallati, F. Longo, G. Merlino, Companion fog computing: Supporting things mobility through container migration at the edge, in: 2018 IEEE International Conference on Smart Computing, 2018, pp. 97–105, http://dx.doi.org/10.1109/SMARTCOMP.2018.00079.

[19] P. Habibi, S. Baharlooei, M. Farhoudi, S. Kazemian, S. Khorsandi, Virtualized SDN-based end-to-end reference architecture for fog networking, in: IEEE International Conference on Advanced Information Networking and Applications Workshops, 2018, pp. 61–66, http://dx.doi.org/10.1109/WAINA.2018.00064.

[20] M.S. de Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner, D. Nehls, O. Keil, F. Schreiner, A service orchestration architecture for fog-enabled infrastructures, in: Second International Conference on Fog and Mobile Edge Computing, 2017, pp. 127–132, http://dx.doi.org/10.1109/FMEC.2017.7946419.

[21] R. Mahmud, F.L. Koch, R. Buyya, Cloud-fog interoperability in IoT-enabled healthcare solutions, in: International Conference on Distributed Computing and Networking, 2018, pp. 32:1–32:10, http://dx.doi.org/10.1145/3154273.3154347.

[22] Q. Qi, F. Tao, A smart manufacturing service system based on edge computing, fog computing, and cloud computing, IEEE Access 7 (2019) 86769–86777, http://dx.doi.org/10.1109/ACCESS.2019.2923610.

[23] N.K. Giang, M. Blackstock, R. Lea, V.C. Leung, Developing IoT applications in the fog: A distributed dataflow approach, in: IEEE International Conference on the Internet of Things, 2015, pp. 155–162.

[24] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin, E. Marin-Tordera, Data preservation through Fog-to-Cloud (F2C) data management in smart cities, in: IEEE International Conference on Fog and Edge Computing, 2018, pp. 1–9.

[25] R. Casadei, M. Viroli, Coordinating computation at the edge: a decentralized, self-organizing, spatial approach, in: IEEE International Conference on Fog and Mobile Edge Computing, 2019, pp. 60–67.

[26] A. Rabay'a, E. Schleicher, K. Graffi, Fog computing with p2p: Enhancing fog computing bandwidth for iot scenarios, in: International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2019, pp. 82–89.

[27] V. Karagiannis, S. Schulte, Comparison of alternative architectures in fog computing, in: 4th IEEE International Conference on Fog and Edge Computing, 2020, pp. 19–28.

[28] J. Bellendorf, Z.Á. Mann, Classification of optimization problems in fog computing, Future Gener. Comput. Syst. 107 (2020) 158–176, http://dx.doi.org/10.1016/j.future.2020.01.036.

[29] C. Hong, B. Varghese, Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms, ACM Comput. Surv. 52 (2019) 97:1–97:37, http://dx.doi.org/10.1145/3326066.

[30] H.R. Arkian, A. Diyanat, A. Pourkhalili, MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications, J. Netw. Comput. Appl. 82 (2017) 152–165.

[31] O. Skarlat, V. Karagiannis, T. Rausch, K. Bachmann, S. Schulte, A framework for optimization, service placement, and runtime operation in the fog, in: 11th IEEE/ACM International Conference on Utility and Cloud Computing, 2018, pp. 164–173.

[32] C. Zhu, J. Tao, G. Pastor, Y. Xiao, Y. Ji, Q. Zhou, Y. Li, A. Ylä-Jääski, Folo: Latency and quality optimized task allocation in vehicular fog computing, IEEE Internet Things J. 6 (2019) 4150–4161, http://dx.doi.org/10.1109/JIOT.2018.2875520.

[33] L. Gu, D. Zeng, S. Guo, A. Barnawi, Y. Xiang, Cost efficient resource management in fog computing supported medical cyber-physical system, IEEE Trans. Emerg. Top. Comput. 5 (2017) 108–119, http://dx.doi.org/10.1109/TETC.2015.2508382.

[34] D. Zeng, L. Gu, S. Guo, Z. Cheng, S. Yu, Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system, IEEE Trans. Comput. 65 (2016) 3702–3712.

[35] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, C. Liu, Fog computing for energy-aware load balancing and scheduling in smart factory, IEEE Trans. Ind. Inf. 14 (2018) 4548–4556.

[36] Y. Wang, K. Wang, H. Huang, T. Miyazaki, S. Guo, Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications, IEEE Trans. Ind. Inf. 15 (2019) 976–986, http://dx.doi.org/10.1109/TII.2018.2883991.

[37] B. Jennings, R. Stadler, Resource management in clouds: Survey and research challenges, J. Netw. Syst. Manage. 23 (2015) 567–619.

[38] P. Pop, M.L. Raagaard, M. Gutiérrez, W. Steiner, Enabling fog computing for industrial automation through Time-Sensitive Networking (TSN), IEEE Commun. Stand. Mag. 2 (2018) 55–61, http://dx.doi.org/10.1109/MCOMSTD.2018.1700057.

[39] M.L. Raagaard, P. Pop, M. Gutiérrez, W. Steiner, Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing, in: 2017 IEEE Fog World Congress (FWC), 2017, pp. 1–6.

[40] K. Fizza, N. Auluck, A. Azim, Improving the schedubality of real-time tasks using fog computing, IEEE Trans. Serv. Comput. (2020).

[41] E. Gomes, M.A.R. Dantas, P.D.M. Plentz, A real-time fog computing approach for healthcare environment, in: F. Xhafa, F. Leu, M. Ficco, C. Yang (Eds.), 13th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Springer, 2018, pp. 85–95, http://dx.doi.org/10.1007/978-3-030-02607-3_8.

[42] H. Kopetz, S. Poledna, In-vehicle real-time fog computing, in: 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop, 2016, pp. 162–167.

[43] Z. Ning, J. Huang, X. Wang, Vehicular fog computing: Enabling real-time traffic management for smart cities, IEEE Wirel. Commun. 26 (2019) 87–93, http://dx.doi.org/10.1109/MWC.2019.1700441.

[44] P. Verma, S.K. Sood, Fog assisted-IoT enabled patient health monitoring in smart homes, IEEE Internet Things J. 5 (2018) 1789–1796, http://dx.doi.org/10.1109/JIOT.2018.2803201.

[45] C. Alcaraz, Secure interconnection of IT-OT networks in Industry 4.0, in: Critical Infrastructure Security and Resilience: Theories, Methods, Tools and Technologies, Springer, 2019, pp. 201–217, http://dx.doi.org/10.1007/978-3-030-00024-0_11.

[46] W. Steiner, S. Poledna, Fog computing as enabler for the Industrial Internet of Things, Elektrotech. Inf.tech. 133 (2016) 310–314, http://dx.doi.org/10.1007/s00502-016-0438-2.

[47] S.V. Gogouvitis, H. Mueller, S. Premnadh, A. Seitz, B. Bruegge, Seamless computing in industrial systems using container orchestration, Future Gener. Comput. Syst. 109 (2020) 678–688, http://dx.doi.org/10.1016/j.future.2018.07.033.

[48] H. Mueller, S.V. Gogouvitis, A. Seitz, B. Bruegge, Seamless computing for industrial systems spanning cloud and edge, in: IEEE International Conference on High Performance Computing & Simulation, 2017, pp. 209–216, http://dx.doi.org/10.1109/HPCS.2017.40.

[49] S. Meixner, D. Schall, F. Li, V. Karagiannis, S. Schulte, K. Plakidas, Automatic application placement and adaptation in cloud-edge environments, in: IEEE International Conference on Emerging Technologies and Factory Automation, 2019, pp. 1001–1008, http://dx.doi.org/10.1109/ETFA.2019.8869256.

[50] M. Aazam, S. Zeadally, K.A. Harras, Deploying fog computing in Industrial Internet of Things and Industry 4.0, IEEE Trans. Ind. Inf. 14 (2018) 4674–4682, http://dx.doi.org/10.1109/TII.2018.2855198.

[51] R. Colelli, S. Panzieri, F. Pascucci, Securing connection between IT and OT: the Fog Intrusion Detection System prospective, in: IEEE Workshop on Metrology for Industry 4.0 and IoT, 2019, pp. 444–448, http://dx.doi.org/10.1109/METROI4.2019.8792884.
[52] L. Li, K. Ota, M. Dong, Deep learning for smart industry: Efficient manufacture inspection system with fog computing, IEEE Trans. Ind. Inf. 14 (2018) 4665–4673, http://dx.doi.org/10.1109/TII.2018.2842821.
[53] P. O'donovan, C. Gallagher, K. Bruton, D.T. O'Sullivan, A fog computing industrial cyber-physical system for embedded low-latency machine learning Industry 4.0 applications, Manuf. Lett. 15 (2018) 139–142.
[54] T.M. Fernández-Caramés, P. Fraga-Lamas, M. Suárez-Albela, M. Vilar-Montesinos, A fog computing and cloudlet based augmented reality system for the Industry 4.0 Shipyard, Sensors 18 (2018) http://dx.doi.org/10.3390/s18061798.
[55] Z. Zhou, J. Hu, Q. Liu, P. Lou, J. Yan, W. Li, Fog computing-based cyber-physical machine tool system, IEEE Access 6 (2018) 44580–44590, http://dx.doi.org/10.1109/ACCESS.2018.2863258.
[56] FORA ETN project, Deliverable D5—Fog infrastructure reference architecture document, 2020, (accessed July 1, 2020). URL: http://www.fora-etn.eu/deliverables/.
[57] P.H. Feiler, B. Lewis, S. Vestal, The SAE avionics architecture description language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering, Technical Report, ARMY AVIATION AND MISSILE COMMAND REDSTONE ARSENAL AL, 2003.
[58] S.A. Team, et al., An Extensible Open Source AADL Tool Environment (OSATE), Software Engineering Institute, 2006.
[59] Y. Wang, D. Ma, Y. Zhao, L. Zou, X. Zhao, An aadl-based modeling method for arinc653-based avionics software, in: IEEE 35th Annual Computer Software and Applications Conference, 2011, pp. 224–229.
[60] V. Gavrilut, B. Zarrin, P. Pop, S. Samii, Fault-tolerant topology and routing synthesis for ieee time-sensitive networking, in: Proceedings of the 25th International Conference on Real-Time Networks and Systems, 2017, pp. 267–276.
[61] M. Barzegaran, B. Zarrin, P. Pop, Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming, in: 2nd Workshop on Fog Computing and the IoT, 2020c, pp. 3:1–3:9.
[62] D. Steinberg, F. Budinsky, E. Merks, M. Paternostro, EMF: Eclipse Modeling Framework, Pearson Education, 2008.
[63] TTTech Computertechnik AG, Nerve, 2020, (accessed July 1, 2020). https://www.tttech-industrial.com/products/nerve/.
[64] K. Sandström, A. Vulgarakis, M. Lindgren, T. Nolte, Virtualization technologies in embedded real-time systems, in: IEEE 18th Conference on Emerging Technologies & Factory Automation, 2013, pp. 1–8.
[65] R. Kaiser, S. Wagner, The pikeos concept: History and design, SysGO AG White Paper, 2007, Available: http://www.sysgo.com.
[66] A. Stanford-Clark, H.L. Truong, MQTT for Sensor Networks (MQTT-SN) Protocol Specification, International business machines (IBM) Corporation Version 1, 2, 2013.
[67] Z. Shelby, K. Hartke, C. Bormann, The constrained application protocol (CoAP), 2014.
[68] P. Danielis, J. Skodzik, V. Altmann, E.B. Schweissguth, F. Golatowski, D. Timmermann, J. Schacht, Survey on real-time communication via ethernet in industrial automation environments, in: Proceedings of the IEEE Emerging Technology and Factory Automation, 2014, pp. 1–8.
[69] M. Schoeberl, P. Schleuniger, W. Puffitsch, F. Brandner, C.W. Probst, S. Karlsson, T. Thorn, Towards a time-predictable dual-issue microprocessor: The patmos approach, in: Workshop on Bringing Theory to Practice: Predictability and Performance in Embedded Systems, OASICS, 2011, pp. 11–21.
[70] S.S. Craciunas, R.S. Oliver, T. AG, An overview of scheduling mechanisms for time-sensitive networks, in: Proceedings of the Real-Time Summer School LÉcole dÉté Temps Réel, 2017, pp. 1551–3203.
[71] T. Adame, M. Carrascosa, B. Bellalta, Time-sensitive networking in IEEE 802.11be: On the way to low-latency WiFi 7, 2019, arXiv preprint arXiv:1912.06086.
[72] 3rd Generation Partnership Project (3GPP), 5G system (5GS); Time-Sensitive Networking (TSN) Application Function (AF) to Device-Side TSN Translator (DS-TT) and Network-Side TSN Translator (NW-TT) protocol aspects; Stage 3, 2019, Release 16.
[73] A. Rajandekar, B. Sikdar, A survey of MAC layer issues and protocols for machine-to-machine communications, IEEE Internet Things J. 2 (2015) 175–186.
[74] M. Vaezi, R. Schober, Z. Ding, H.V. Poor, Non-orthogonal multiple access: Common myths and critical questions, IEEE Wirel. Commun. 26 (2019) 174–180.
[75] V. Karagiannis, Compute node communication in the fog: Survey and research challenges, in: Workshop on Fog Computing and the IoT, 2019, pp. 36–40.
[76] M. Chiang, S. Ha, I. Chih-Lin, F. Risso, T. Zhang, Clarifying fog computing and networking: 10 questions and answers, IEEE Commun. Mag. 55 (2017) 18–20, http://dx.doi.org/10.1109/MCOM.2017.7901470.
[77] B. Alt, M. Weckesser, C. Becker, M. Hollick, S. Kar, A. Klein, R. Klose, R. Kluge, H. Koeppl, B. Koldehofe, W.R. KhudaBukhsh, M. Luthra, M. Mousavi, M. Mühlhäuser, M. Pfannemüller, A. Rizk, A. Schürr, R. Steinmetz, Transitions: A protocol-independent view of the future internet, Proc. IEEE 107 (2019) 835–846, http://dx.doi.org/10.1109/JPROC.2019.2895964.
[78] R.S. Crespí, A. Armentia, I. Sarachaga, O. Casquero, F. Pérez, M. Marcos, OPC UA aggregation server in the fog, in: 24th IEEE International Conference on Emerging Technologies and Factory Automation, 2019, pp. 1256–1260, http://dx.doi.org/10.1109/ETFA.2019.8869358.
[79] G. Pardo-Castellote, OMG data-distribution service: Architectural overview, in: 23rd International Conference on Distributed Computing Systems Workshops, 2003, pp. 200–206, http://dx.doi.org/10.1109/ICDCSW.2003.1203555.
[80] C. Pahl, A. Brogi, J. Soldani, P. Jamshidi, Cloud container technologies: a state-of-the-art review, IEEE Trans. Cloud Comput. (2017).
[81] I. Mavridis, H.D. Karatza, Combining containers and virtual machines to enhance isolation and extend functionality on cloud computing, Future Gener. Comput. Syst. 94 (2019) 674–696, http://dx.doi.org/10.1016/j.future.2018.12.035.
[82] M. Barzegaran, A. Cervin, P. Pop, Performance optimization of control applications on fog computing platforms using scheduling and isolation, IEEE Access 8 (2020) 104085–104098.
[83] M. Kadar, S. Tverdyshev, G. Fohler, System calls instrumentation for intrusion detection in embedded mixed-criticality systems, in: 4th International Workshop on Security and Dependability of Critical Embedded Real-Time Systems, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
[84] C. Avasalcai, S. Dustdar, Latency-aware distributed resource provisioning for deploying IoT applications at the edge of the network, in: Advances in Information and Communication, Springer International Publishing, Cham, 2020, pp. 377–391.
[85] V. Karagiannis, N. Desai, S. Schulte, S. Punnekkat, Addressing the node discovery problem in fog computing, in: 2nd Workshop on Fog Computing and the IoT, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
[86] V. Karagiannis, S. Schulte, J. Leitão, N. Preguiça, Enabling fog computing using self-organizing compute nodes, in: IEEE 3rd International Conference on Fog and Edge Computing, 2019, pp. 1–10.
[87] O.M. Group, About the OPC-UA/DDS gateway specification, 2020, (accessed July 1, 2020). https://www.omg.org/spec/DDS-OPCUA/About-DDS-OPCUA/.
[88] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.E. Arzen, V. Romero, C. Scordino, Resource management on multicore systems: The actors approach, IEEE Micro 31 (2011) 72–81.
[89] M. Cheminod, L. Durante, A. Valenzano, Review of security issues in industrial networks, IEEE Trans. Ind. Inf. 9 (2013) 277–293, http://dx.doi.org/10.1109/TII.2012.2198666.
[90] S. Yi, Z. Qin, Q. Li, Security and privacy issues of fog computing: A survey, in: K. Xu, H. Zhu (Eds.), Wireless Algorithms, Systems, and Applications, Springer International Publishing, Cham, 2015, pp. 685–695.
[91] McKinsey & Company, Industry 4.0 How to navigate digitization of the manufacturing sector, 2020, (accessed July 1, 2020). https://www.mckinsey.de/files/mck_industry_40_report.pdf.
[92] T. Adali, D.J. Miller, K.I. Diamantaras, J. Larsen, Trends in machine learning for signal processing [in the spotlight], IEEE Signal Process. Mag. 28 (2011) 193–196, http://dx.doi.org/10.1109/MSP.2011.942319.
[93] C. Pahl, Containerization and the PaaS cloud, IEEE Cloud Comput. 2 (2015) 24–31, http://dx.doi.org/10.1109/MCC.2015.51.
[94] E. Lear, R. Droms, D. Romascanu, RFC 8520: Manufacturer usage description specification, in: Internet Engineering Task Force, 2019, March.
[95] N. Desai, S. Punnekkat, Enhancing fault detection in time sensitive networks using machine learning, in: 12th International Conference on COMmunication Systems & NETworkS, 2020, pp. 714–719.
[96] J. Qian, S. Sengupta, L.K. Hansen, Active learning solution on distributed edge computing, 2019, arXiv preprint arXiv:1906.10718.
[97] J. Qian, S.P. Gochhayat, L.K. Hansen, Distributed active learning strategies on edge computing, in: 6th IEEE International Conference on Cyber Security and Cloud Computing/5th IEEE International Conference on Edge Computing and Scalable Cloud, 2019, pp. 221–226.
[98] J. Qian, L.K. Hansen, X. Fafoutis, P. Tiwari, H.M. Pandey, Robustness analytics to data heterogeneity in edge computing, Comput. Commun. 164 (2020) 229–239.
[99] G.C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. Vol. 24, Springer Science & Business Media, 2011.
[100] I. Boldea, S.A. Nasar, Electric Drives, CRC press, 2016.

[101] N.R. Storey, Safety Critical Computer Systems, Addison-Wesley Longman Publishing Co., Inc, 1996.
[102] A. Cervin, P. Pazzaglia, M. Barzegaran, R. Mahfouzi, Using JitterTime to analyze transient performance in adaptive and reconfigurable control systems, in: Proc. of IEEE International Conference on Emerging Technologies and Factory Automation, 2019, pp. 1025–1032.
[103] M. Barzegaran, V. Karagiannis, C. Avasalcai, P. Pop, S. Schulte, S. Dustdar, Towards extensibility-aware scheduling of industrial applications on fog nodes, in: Proc. of 4th IEEE International Conference on Edge Computing, 2020b, pp. 1–9.
[104] N. Reusch, S.S. Craciunas, P. Pop, Towards Security-Aware Scheduling for TSN-Based Distributed Cyber-Physical Systems, Technical Report, Technical University of Denmark, 2020.