





Article

A Framework for Service-Oriented Architecture (SOA)-Based IoT Application Development

Joao Giao ^{1,*}, Artem A. Nazarenko ^{1,2}, Fernando Luis-Ferreira ¹, Diogo Gonçalves ³ and Joao Sarraipa ¹

¹ Faculty of Sciences and Technology & UNINOVA-CTS, Nova University of Lisbon, 2829-516 Monte Caparica, Portugal

² Chair of IT Security, Brandenburg University of Technology Cottbus-Senftenberg, Konrad-Wachsmann-Allee 5, 03046 Cottbus, Germany

³ School of Economics, University of Bristol, Priory Road Complex, Priory Road, Bristol BS8 1TU, UK

* Correspondence: jgs@uninova.pt

Abstract: In the last decades, the increasing complexity of industrial information technology has led to the emergence of new trends in manufacturing. Factories are using multiple Internet of Things (IoT) platforms to harvest sensor information to improve production. Such a transformation contributes to efficiency growth and reduced production costs. To deal with the heterogeneity of the services within an IoT system, Service-Oriented Architecture (SOA) is referred to in the literature as being advantageous for the design and development of software to support IoT-based production processes. The aim of SOA-based design is to provide the leverage to use and reuse loosely coupled IoT services at the middleware layer to minimise system integration problems. We propose a system architecture that follows the SOA architectural pattern and enables developers and business process designers to dynamically add, query or use instances of existing modular software in the IoT context. Furthermore, an analysis of utilization of modular software that presents some challenges and limitations of this approach is also in the scope of this work.



Citation: Giao, J.; Nazarenko, A.A.; Luis-Ferreira, F.; Gonçalves, D.; Sarraipa, J. A Framework for Service-Oriented Architecture (SOA)-Based IoT Application Development. *Processes* **2022**, *10*, 1782. <https://doi.org/10.3390/pr10091782>

Academic Editors: Shahryar Sorooshian and Madjid Tavana

Received: 29 July 2022

Accepted: 27 August 2022

Published: 5 September 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: application programming interfaces; Internet of Things; interoperability; middleware; modular construction

1. Introduction

The Internet of Things (IoT) departs from the conceptualization that everyday objects have internet connection and are equipped with sensing, actuation and processing capabilities so that such devices will provide benefits both in terms of new services and in terms of efficiency improvement for existing services and processes [1]. The IoT is considered a large-scale system that allows billions of physical objects to communicate with each other and collect data over the internet [2]. In this regard, the concept of “things”—devices with sensing and computing capabilities that communicate with other devices/things, bringing new functionalities and added value into many application domains—has appeared [3].

IoT systems bridge the cyber and the physical space and thus rely on a wide range of both hardware and software solutions. In the case of hardware, several challenges have been identified to satisfy IoT-immanent demands, such as mobility and autonomy, energy-use optimization and decreased production costs [4]. The growing number of internet-connected devices directly affects the amount of data generated to be potentially used by different applications to create value, which, in turn, influences common approaches to data treatment and management. Let us consider, for example, the usage of data in the transport and logistics domain, where vehicles such as cars and buses and infrastructure objects such as roads and highways are equipped with sensors that have the processing capability to transmit data between each other in real-time. This would allow accurate estimation of traffic density, allowing users to select the quickest route to their destination based on the number of road users at that point in time. In addition, it is possible to make

inferences such as the amount of gas or electricity required for a specific vehicle to reach a specific destination based on the traffic density in real-time.

To correctly identify the most-promising areas where IoT technologies are able to have the greatest impact, it is necessary to evaluate the place of IoT technologies in the global economy. For instance, Manyika et al. [5] estimated the impact of IoT technologies at around \$11.1 trillion by 2025, a value that is equivalent to 11 percent of the world economy. As Xie and Chen [6] noted, the introduction of an IoT-based supply chain and logistics management reduces the Average Queue Time of vehicles by approximately seven times their initial value, improving the efficiency and cooperation of various departments within a business and minimising supply-chain disruptions, which cause extensive costs to businesses. As a result, the relationship between consumers and suppliers is optimised, and the overall competitiveness of the business is improved, highlighting the importance of developing and implementing these IoT technologies at the micro and macro levels.

Furthermore, benefits of the IoT are observed in the business supply chains profiting from the real-time data that are further analysed to better understand supply and demand patterns for goods and services. This allows accurate analysis of surpluses or shortages of specific goods. It also ensures that producers align their pricing strategies and production resources to manage their stock levels based on current demand. An IoT system may combine heterogeneous communication interfaces from different equipment vendors or the coexisting differences in communication specifications [7]. In this regard, serious obstacles arise on the way to the establishment of interoperability between a myriad of potentially connected IoT devices, as there is no well-defined reference standard for IoT platforms, and the likelihood of one appearing in the near future is low. Hence, there is a necessity to find a strategy to tackle the lack of interoperability, which is a known cause for technological incompatibilities when integrating new applications from different sources. An additional challenge is the flexibility required for “plug-and-play” deployment of new IoT devices to an existing IoT environment. The same is relevant for interconnection of different IoT platforms [8]. To overcome this challenge, attempts at addressing IoT interoperability through standardization activities have been undertaken [9]. In this sense, the authors of [10] prove that Service-Oriented Architecture (SOA) [11] is one of the most suitable architectural paradigms to achieve interoperability in IoT systems. By definition, interoperability is the ability of two or more systems or components to exchange information and use the information that has been exchanged [12]. In [11], Marks et al. define SOA as “a conceptual business architecture where business functionality or application logic is made available to SOA users or consumers, as shared, reusable services with exposed interfaces”. In this regard, the challenge is to pursue a strategy to support IoT interoperability by introducing a SOA-based conceptual business architecture to orchestrate the interactions between IoT devices within different platforms.

To solve the integration and interoperability problems of digital technological environments, the present research proposes an approach based on the use of software technology designed for online computing, thus answer to the following research question: How can existing software modules be integrated in an industrial factory with an existing system without changing the current system’s architecture? The main contribution from this article is a framework that facilitates the integration of different software modules while using multiple services and/or different communication interfaces. It also allows the reuse of these software modules in distinct scenarios in distinct domains. Therefore, the presented research can be of use to facilitate the integration of existing IoT services and to improve interoperability between IoT components.

When using IoT devices and services throughout global connectivity and accessibility, security becomes a critical aspect. In fact, there is a risk that anyone may try to access the IoT devices from anywhere at anytime, thus exposing them to cyberattacks against the IoT applications and networks [13]. Therefore, this paper also considers related security and safety issues.

This paper presents in Section 2 an overview of interoperability, SOA and security concerns in the context of Industrial IoT (IIoT). In Section 3, we describe the solution and explain the developed framework. Section 4 introduces a demonstration scenario as well as the developed use case with different software modules. Finally, the results are presented in Section 5, and Section 6 is a discussion of the insights from this work.

2. Related Work

Nowadays, the Industry 4.0 paradigm, along with the tight integration of various Cyber Physical and IoT systems, focuses on interoperability and secure interconnection among heterogeneous components of a system [14]. This section discusses relevant topics related to integration and interoperability, and how SOA design can promote the seamless integration of heterogeneous services.

2.1. IoT-Driven Integration

IoT plays a central role in Industry 4.0. As a matter of fact, IIoT is the employment of IoT in industry, where sensors are embedded in all the components related to a manufacturing process [15]. These devices collect data that can be used to monitor and control the factory's processes, helping its efficiency, productivity or even the safety of the workers in the factory. A wide range of IIoT applications have been released in different areas, with some notable examples in economic, energy, engineering and industrial control, instrumentation, manufacturing and transportation [16].

IoT is shifting how systems sense and acquire information from the environment, resulting in the integration of multiple devices within the same system. In fact, IoT devices are bridging digital and physical worlds. The full potential of IoT will be achieved when everyday things can sense the environment and understand the context of the events occurring within this environment to jointly act in collaboration with other things. Thereby, IoT emphasizes interaction among the networked things to generate added value [3]. Moreover, users can access the information from IoT devices through the Internet, getting notified and becoming empowered to take action to control the environment [17].

An IoT system requires some aspects/properties to be considered at the design phase to enhance the advantages of interconnected objects in a network [13,18], such as:

- *Adaptivity*: IoT devices are often mobile, moving across wide geographical area, while still needing to continue interacting with other assets in real-time [17]. For this reason, communication devices should adjust themselves in new locations and collaborate with the local things.
- *Context-awareness*: Systems and their components need to understand the context of events to accurately adapt their actions to satisfy the users' needs and requirements and thus enrich the available services.
- *Autonomy*: Applications/IoT devices should be able to manage spontaneous interactions when they move to new locations and get in other objects' communication range [19]. These interactions should occur without or with minimal human involvement.
- *Distributivity*: IoT devices are intrinsically distributed, resource-constrained components and often require middleware to function properly [20]. The distributive approach helps reduce the overhead of centralized architectures, since IoT systems have a vast number of devices that are constantly collecting and exchanging data over the internet.
- *Interoperability*: In an IoT environment, heterogeneous devices, technologies and applications should operate to achieve the goal of integration, a property of utmost interest in Industry 4.0. Interoperability has the potential to make software components or systems accessible, manageable and potentially linked despite their differences in interface, execution platform and language [21].

Interoperability between different devices and applications becomes a major challenge in IoT. The reasons range from different design patterns to the existence of a wide set of

technologies, both old and new, including a large number of incompatible communication protocols.

2.2. Interoperability in IoT

Interoperability has multiple dimensions to be considered. These dimensions have to be addressed through different strategies that can assist in overcoming the interoperability gaps and platform connectivity limitations to enhance collaboration among heterogeneous applications and systems.

Interoperability among heterogeneous systems can occur in the following dimensions [8]:

- *Technical Interoperability*: hardware/software components, platforms and systems that allow machine-to-machine communication;
- *Syntactical Interoperability*: related to data formats;
- *Semantic Interoperability*: related to the meaning of contents.

These are the dimensions that an IoT platform has to handle to correctly exchange data and pass information to other modules. An interoperable platform provides organizations with the capability to exchange data across systems that rely on heterogeneous infrastructures and ensures that these data are correctly interpreted.

The number of IoT platforms is constantly growing. However, most of these platforms are narrowly focused on specific domains or applications [18]. Most of those platforms were created for a specific domain or application, which results in a fragmented IoT landscape with many vertical IoT environments, which are rarely interconnected [18]. This contradicts the basic IoT principle of interconnectivity, regardless of whether the platforms are physically connected. This non-uniformity is an obstacle when application developers aim to establish cross-platform and cross-domain IoT solutions [9]. Moreover, some architectures are designed based on existing IoT standards. However, the problem appears with the growing number of standards that are not always compatible. This problem is also caused by quick evolution of technology and the emergence of new hardware and software paradigms [22].

Cross-platform IoT interaction is necessary when applications need to access different IoT platforms to utilize data made available by them. To achieve this goal, developers need extensive knowledge of each platform's specific Application Programming Interfaces (APIs) and information models [9]. In line with this, the European Commission and other international organizations have already developed domain-specific, potentially open IoT platforms (some can be found in [8]) to facilitate such cross-platform application development. Some examples of these IoT platforms are the IOT-A (<https://cordis.europa.eu/project/id/257521>), (accessed on 28 July 2022) an architectural reference model for IoT, or Butler (<https://cordis.europa.eu/project/id/287901>), (accessed on 28 July 2022) a platform that integrates current IoT technologies and development processes. Another example is the FIWARE platform initiative, which provides a set of open-source modular components called *enablers*. FIWARE has the purpose of creating an open-source cloud platform while boosting development and adoption of Future Internet technologies in Europe [23] that would help developers to create software applications using different software components that provide different functionalities. FIWARE allows users to deploy enablers that are compliant with the FIWARE Context Broker, which is used to manage context information of the platform or solution. These enablers can be installed and configured only through command line interface using Docker or Kubernetes functionalities [24]. These enablers provide services to be used by software developers in different domains, such as manufacturing, health or energy.

It is thus far a fact that traditional IoT lacks uniform standardization such as that found in communication protocols and sensing technology [25], and as a result, it can be a challenge to ensure high levels of interoperability between all IoT modules. Moreover, in this paper, we identify and address several IoT middleware-related challenges, namely: (i) resource or service discovery that includes more than just simple IP address discovery,

but also considers services semantics; (ii) security, privacy and trust mechanisms utilizing authentication, encryption and access control [26]; and (iii) scalability of resource management, resilience and on demand utilization, presuming that even if one of the software modules fails, the system quickly adapts without significantly affecting user request processing [27]. The next subsection presents an architecture standard that has been used to cope with these IoT integration gaps.

2.3. Service-Oriented Architecture and IoT

SOA is an architectural style that uses a set of services in order to build complex systems of systems [28] in which standalone applications and services can be linked to other applications. SOA has been used in the industry and validated by many researchers as a promising solution to support a variety of standards by the exposure of one or more interfaces. The usage of loose software modules and its high flexibility can open new horizons for business purposes and revolutionize the manufacturing processes [29]. These software components, referred to as “services”, define independent units of software modules, or a set of functionalities, responsible for specific tasks. They contain the code and the necessary information to be used by the system in order to do a complete job [30], which can be reusable by third-party applications and platforms without knowledge of how they are implemented [31]. Thus, in SOA-based IoT systems, each device can have the role as service consumer as well as the role of a service provider, interacting via compatible APIs. This type of architecture allows systems to publish, discover or select a wide range of services provided by independently deployed IoT devices to communicate and be used by multiple applications across the system [32]. Additionally, SOA-driven systems have the characteristic of *service adaptivity*, meaning heterogeneous services should be able to interoperate without needing modifications.

In general, the most significant benefits from using SOA are the reuse of technology and the agility to integrate new devices and modules in a “plug-and-play” manner according to the system’s needs [33,34]. The reuse and maintenance of software modules enables, by using small, independent interconnected services instead of complex monoliths, multiple added values in general IT and cloud contexts: service reutilization, reduced complexity, faster testing cycles and agile development due to fewer dependencies code-wise, and less support required as a consequence of fewer bugs [34]. Thus, SOA’s goal is to allow such a platform to link IoT applications and their services by uniform and structured formats, as well as to enable abstraction of underlying implementation complexity. A systematic literature review of SOA is available in [35], and a comprehensive analysis of security on SOA-based IoT middleware systems is available in [36]. Similar to SOA, another service-based architecture is Microservice Architecture (MSA). Both MSA and SOA are generally distributed architectures that lend themselves to more loosely coupled and modular applications by using service functionality. However, they represent different architectural styles; for example, microservices are small, fine-grained services, while services in SOA range in size from very small to large enterprise services. They also differ concerning data sharing, as MSA uses a style where microservices share as little data as possible, whereas SOA encourages the opposite concept of sharing as much data as possible [37]. A thorough comparison between SOA and MSA is available in [38].

Zhu et al. [39] developed an ontology model that correlates IoT services with the physical objects within the system. To use the developed semantic model, all IoT objects must be defined within an ontology. Uviase et al. [40] presented theoretical research focused on using an SOA design type to develop a highly scalable, extensible and fault-tolerant integration framework to integrate IoT devices and systems. The main message that can be extracted from the literature is that SOA can be used to ensure interoperability in the IoT domain.

Different formal models and standards already exist within IoT that enable easy integration and uniformization across IoT resources, such as W3C Semantic Sensor Network Ontology (SSN) [41] and Web of Things (WoT) [42]; ISO/IEC 30161:2020–Requirements

of IoT data exchange platform for various IoT services or European Telecommunications Standards Institute [43]; and European Telecommunications Standards Institute (ETSI) Smart Applications REference ontology [44]. However, there is no common agreement on which standard must be used for all IoT platforms to describe their services, in contrast to, for instance, web services [45]. This poses a great challenge for integration of existing services to be used by smart applications. Furthermore, according to the authors of [11], service-level metrics may be used to increase the success of an SOA application for IoT.

The presented framework can act as middleware, providing the necessary abstraction layer to expose the IoT resources through a predefined REST API. In the context of this work, middleware is a software component that integrates heterogeneous computing and communication devices, and supports interoperability within different applications and services running these devices. It abstracts the complexities of the system or hardware and hides the heterogeneous interfaces. A general analysis of middleware within the IoT domain can be found in [46–48].

2.4. Related Middleware Platforms

As mentioned before, IoT is considered a large-scale system that uses intelligent and smart sensors and actuators that interact autonomously with minimal human intervention and is connected to the internet. To this end, to have successful communication between all IoT devices, they need to use a common language. As stated in a previous subsection, this interoperability problem is difficult to solve through agreement on a universal standard. For this reason, middleware can be used since it can provide interoperability among incompatible devices and applications [49]. In the following, we summarize and discuss the main characteristics of the most-relevant IoT middleware platforms considered for this study, and a comparison of them is made in Table 1.

- Orion: Open-source component that allows the user to manage context information, including update context, queries context, registrations and subscriptions. Developed by the FIWARE foundation, it can be useful for providing functionalities for context management and can be useful in use cases that involve data brokers between producers (e.g., sensors) and consumers (e.g., smartphone applications). It has an NGSI interface that allows users to make several operations, such as register, update and query context information, as well as receive notifications when changes in context information take place (e.g., a sensor value has changed). This component has a Metrics API that provides a few statistical data about their services. Orion also implements a simple multitenant/multiservice model that ensures entities from one service cannot be affected by other services [50].
- ETSI M2M: Set of standards provided by ETSI defining the entities and functions to enable interoperability between M2M services. The ETSI M2M architecture is agnostic to underlying networks and includes two basic elements: Service Capability Layers and Service Capabilities. The first functionality exposes the Service Capabilities to M2M applications and should be implemented on top of devices and/or gateways [51]. The second functionality provides generic M2M functions, analogous to GEs in FIWARE, including communication management, application management, device discovery and integration, etc. It also supports both the request–response and publish–subscribe communication models [52]. According to the authors of [53], ETSI M2M has no reference implementation, compared to, for instance, FIWARE. However, the authors provide their implementation of an ETSI M2M broker and APIs and also benchmark the ETSI M2M implementation against the FIWARE-based solution. ETSI M2M enables connections between physical devices and their digital representations, called Business Applications [54]. However, there is limited information on the possibility of creating multiple instances of Business Applications.
- Ptolemy Accessor Host: Actor-based framework that relies on the notion of an actor as a central element. In the context of this component, “actor” is used to specify a software component that is triggered based on an input event and produces an output

event. One specific type of actor is called an “accessor”, which wraps a device or a service in an actor interface [55]. Every accessor possesses an interface serving as a local proxy and implementation encapsulating the API of a physical device or remote service [26]. Accessors can form applications that are placed within the accessor host, providing registration and discovery functionalities. Moreover, Ptolemy provides user interface with drag-and-drop capabilities to connect actors and accessors. Ptolemy Accessor Host supports instantiation functionality, which allows the creation of multiple accessors within other accessors [56]. To the best of our knowledge, metrics are not available on the Ptolemy platform.

- DeviceHive: Middleware designed to enable message exchange between smart devices and client applications [57] that supports a wide variety of communication models, such as publish/subscribe (through MQTT), Rest and Websockets [58]. It also supports the abstraction or logical grouping of different appliances based on, for instance, their location [59]. Authentication is accomplished using a JSON Web Token. DeviceHive API is a service able to manage several resource types, for instance, the “Network”, which is an isolation entity encapsulating multiple devices. However, the solution does not provide metrics to assess API performance [60].
- NodeRED: Open-source middleware provided by IBM. The main element of the NodeRED environment is called a *Node*, which is the visual representation of a block of Javascript codes providing specific functionalities. Moreover, NodeRED provides a library of different nodes with different functionalities, including, for instance, MQTT nodes. The advantage of this middleware is the visual canvas that is used to drag and drop different elements/nodes and to establish connections to compose IoT applications [26]. A device or service has to possess the API as a node.js library or module accessible by NodeRED to communicate with other devices or services. While good for rapid prototyping, NodeRED does not possess service-discovery capabilities. NodeRED contains a module counts and calculates the metrics per message topic.
- Ignition: Modular server-based middleware that combines the SCADA environment and OPC UA communication [61]. Other modules that can be added on-demand include databases, connectivity gateways, industrial control system security services, charts, alarm dispatchers, etc. According to Inductive Automation—the company that developed this middleware—Ignition has the following distinguishing features: support for some SQL databases, synchronization of design and run-time, cross-platform nature and simple deployment procedure [62]. This platform also enables a wide range of metrics, including a dashboard to visualize statistics. Moreover, this component allows decoupling of physical devices from applications, which enables service isolation [63].

Table 1. Comparison of IoT middleware functionalities.

Name	Communication Interface	Metrics	Plug-and-Play	Instantiation	Installed Service Isolation	User Interface
Orion	NGSI	Yes	Yes	No	Yes	No
ETSI M2M	REST	No	No	N/A	Yes	Option available
Ptolemy Accessor Host	HTTP	No	No	Yes	Yes	Yes
DeviceHive	EST, Websockets, MQTT	No	No	Yes	Yes	No

Table 1. Cont.

Name	Communication Interface	Metrics	Plug-and-Play	Instantiation	Installed Service Isolation	User Interface
NodeRED	HTTP, MQTT, OPC UA	Yes	Yes	No	No	Yes
Ignition	OPA UA, related APIs	Yes	Yes	Yes	Yes	Yes
Proposed Solution	REST, NGSI	Yes	Yes	Yes	Yes	Yes

The horizontal row heading in Table 1 lists the key functionalities for SOA-based middleware components to allow connection with existing IoT modules. Each row of this table enumerates the features of the most-relevant IoT middleware platforms considered for this study, which were discussed in this section. *Communication interface* contains the networking protocols and communication capabilities of the platform, which are used to connect external modules. These protocols use a variety of communication models and can be used for user-centric applications such as home automation, or for messaging purposes between devices and to send/receive messages between sensor nodes [64]. In software development, *Metrics* are measurable characteristics of the analysed software to quantify consistency and quality of an asset and to plan, support progress tracking and identify problems by software architects and developers [65]. *Plug-and-Play*, *Instantiation* and *Installed Service Isolation* refer to the capability to connect to other modules, create multiple instances of the module and specify a network isolation between modules, respectively. These properties are useful in IoT systems as they need to interconnect with multiple devices, and they must adapt to the surrounding environment. *User Interface* facilitates visualization of the module's properties and management. As depicted in Table 1, all platforms lack some of the features we describe. Even though Ignition has the characteristics we seek for integrating existing modular components, it does not connect some components, such as FIWARE enablers, that have NGSI communication protocol, a key requirement for this work.

2.5. Security and Safety Challenges

Since technological developments are emerging at a fast pace, more-tangible approaches must be taken to ensure security and privacy of user data within intelligent systems adopted in business. Correct identification of more-tangible approaches addressing security and privacy challenges of IoT technologies is a necessary step towards the establishment of a more reliable collaborative environment [66]. Given this, adequate cybersecurity measures must be implemented to ensure immunity to the potential threats that will arise in the near future. An additional challenge comes from the complex nature of IoT systems with different security dimensions [67]. For usefulness to be maximized and risk to be minimized, businesses and producers must work together to develop IoT technologies that are secure by default, allowing their usage without endangering the privacy of stakeholders.

The convergence of security and privacy domains is of great importance against intellectual property and identity theft. This allows user authentication and prevents manipulation of data stored on servers and other internet-enabled devices. An additional threat is the use of web-integrated components such as actuators and sensors requiring real-time data transmission that might be susceptible to man-in-the-middle attacks aimed at modifying actuator action, causing damage to the infrastructure [68]. For instance, the application of IoT in supply chain networks and manufacturing brings the ability to make added-value decisions, but with the price of increased risk that sensitive data may be exposed to adversaries if it is created and stored outside the system's premises. Therefore, risk management is needed for protecting critical infrastructure when the architecture of the system has been defined [69].

In this regard, a strong focus is made on data security, as it might affect data-driven operational decision-making, adding complexity to risk-management processes. Businesses must, therefore, remain increasingly vigilant for possible risks attributed to the use of newly introduced technologies that, besides having advantages, could also induce risks and vulnerabilities that could threaten users' data security and privacy [70].

IoT must be considered as a long-term investment for companies. A successful IoT solution must be the one that implements a structured approach by identifying threats and finding appropriate responses reinforcing security measures. Given the high volume of sensitive data exchanged by IoT solutions and the critical requirement to process data in real- or close-to-real-time, security is considered a critical factor for the success of IoT solutions within enterprises.

One particular aspect of a more-tangible approach is privacy-enhancing technology. This includes inter alia, a virtual private network (VPN) that should be established and accessed solely by authorized users. However, the use of a VPN could hinder efficient data exchange between third parties by increasing transfer time. This occurs through a reduction of the routes of convergence, which increases the required time to update the necessary routes for data transfer. The latter is prevalent amongst large enterprises employing Border Gateway Protocol VPNs to ensure that sites in different locations are interconnected with the same level of security [71]. The sensitive nature of connection stability amongst this type of VPNs implies potential bottlenecks as the scale of the network increases, thereby impacting an essential feature of intelligent interconnected systems and IoT solutions, affecting the operation processes of an enterprise.

3. Solution Description

The framework presented in this work is based on an SOA-based conceptual architecture that provides functionalities that allow the integration and uniformization of enablers to be used by software applications. Enablers are open-source software components developed to be easily integrated with third-party applications and providing different functionalities. These modular components execute specific services that can be used in multiple domains. As an example, Lagsaiar et al. [72] developed a system by using existing software modules, an approach that was characterized by simplicity, reliability, low cost and ease of construction. Thus, enablers are software components that provide multiple services, developed either for general or specific usage in different domains, that can be used by smart applications [73]. They can be connected to IoT sensors and actuators or to data storage, acting as middleware to store and retrieve data. As service-oriented software, enablers allow dynamic interaction with real-world devices and applications. These software solutions can use different communication protocols, such as RESTful API, NGSiv1 or NGSiv2 [74]. This same framework, which was designed to integrate different enablers, was developed under the vf-Os project, the overall goal of which was to develop a virtual factory Open Operating System (vf-OS) [75].

SOA-based systems are organized into horizontal technical layers, and their services also follow this approach, being grouped according to their functional similarity (e.g., business services, device services or process services) [10]. The goal of this categorization is to use it as a foundation to increase service characterization granularity. Based on this, the presented work defines two types of enablers based on their maturity level, usage, popularity and date of development

- *Generic Enablers (GE)*: Software components from the FIWARE project. These enablers provide base services that help applications use IoT devices and third-parties applications to process data and media in real-time on a large scale, perform Big-Data analysis or incorporate advanced features for interaction with end-users. Usually, this type of enabler must be approved by specific guidelines and public dissemination through FIWARE channels [76].

- *Specific Enablers*: Set of software modules developed by other researchers to help developers create applications that provide functionalities that are more specific to the system's domain. User instructions and software testing are not standardized.

As mentioned before, there is a diversity of protocols in the IoT domain, and to deal with this challenge, this work presents the Enablers Framework (EF) component, which acts like an interface proxy, allowing applications to use a unique REST API for all registered enablers. This is a common approach in many IoT platforms [17] that allows IoT inter-platform communication through middleware to connect non-interoperable services. The presented framework explores this methodology to ease the integration between REST applications and registered enablers, allowing them to be integrated into SOA-based systems. This approach facilitates the usage of enablers to integrate such diverse solutions during the application development phase. Figure 1 shows a functional architecture of EF.

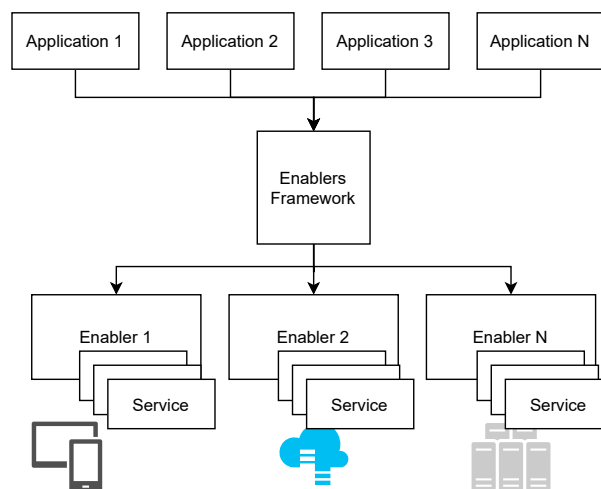


Figure 1. Interoperability between applications and enablers. Each enabler can have different services that can be accessed by different applications only through the EF component.

In Figure 1, the EF is the central point of communication with all system applications and enablers. Several applications can interact with different enablers, each one providing unique services, including data acquisition from sensors. The EF manages the integration of enablers through instantiation of new entities of available services. This enables a protocol-agnostic approach for applications to communicate with IoT devices as well as for replication of existing software components through instantiation. Enablers instantiation allows a system to: (i) have a high level of fault tolerance; in other words, IoT devices can continue to operate in the case of failure of one of the software modules; (ii) accomplish tests on newly added industrial equipment on the fly, i.e., without affecting the production process; (iii) reuse existing software modules for different applications for different domains; and (iv) securely isolate IoT devices from system enablers (e.g., databases, system services, etc.), meaning it is possible to instantiate multiple services on the same machine that are not connected to each other by utilizing Docker, as presented in Figure 2. The figure shows the Docker network architecture used by the EF. Its main goal is to allow communication between registered enablers while isolating them from enabler subcomponents; “efi” is the private network for all EF subcomponents, and “efn” is the network that allows communication between the Request Handler component (presented in next subsection) and the main container of each enabler. Each time an enabler is installed, an internal network (enablerN internal) is created for that enabler, encapsulating the containers for all the enabler’s subcomponents. The creation of a private network for each enabler improves the security of the system, as it separates the internal component(s) of the enabler (e.g., an enabler can have a database, a user interface or a data-filtering component). Thus, the proposed framework operates as an intermediate isolation and protection module for the IoT system.

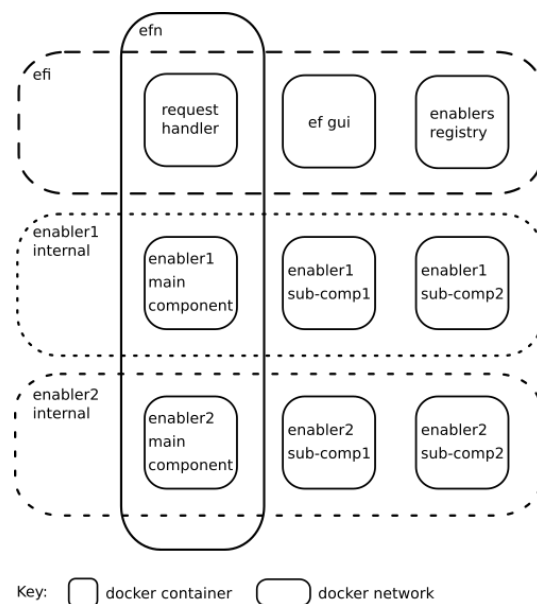


Figure 2. EF instantiation networks. The main enabler module can only be accessed through the Request Handler, an EF submodule.

3.1. Software Architecture

The architecture presented in Figure 3 represents the internal structure and inter-communications of the EF components.

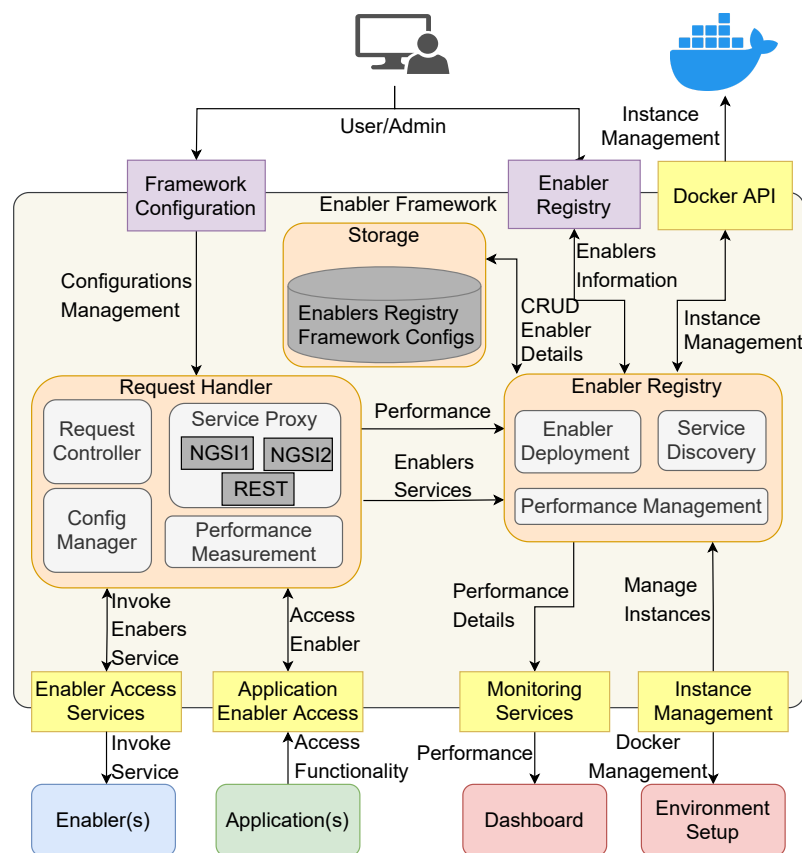


Figure 3. EF internal components and its interactions with external applications and modules.

In Figure 3, the boxes at the bottom represent the system’s components that can interact with the EF. These can be: (i) enablers, which provide the diverse services to be

used by the applications; (ii) a dashboard, which displays metrics based on historical EF data in order to map usage characteristics and track possible errors that might occur during the enabler's run-time; (iii) the environment setup, which can manage the required instances of enablers to be used by the applications; and (iv) the application, which offers the system's functionalities to the final user. The yellow boxes represent the interfaces between EF internal components and external services. These access points have a distinct and straightforward API to simplify their usage by the application developer. The purple boxes represent user interfaces, allowing the administrator (Admin) to configure and access EF data. The black arrows represent the interactions among various components and services. Although these arrows presume different data responses, such as data transfer confirmations or errors, they are not represented on the EF architecture diagram to simplify its representation. The interactions of EF's internal components, represented by orange boxes, need to be in the same environment to be more efficient. The internal components are responsible for the following functionalities:

- *Enabler Registry*: This module allows developers to register a new enabler or to search for an available enabler. Moreover, EF is responsible for providing instruments for performing CRUD (create, read, update and delete) operations. This module also contains a Docker management unit that allows the Admin to install new enabler instances and allows the configuration of various parameters for proper functioning of enablers. Enabler Registry could also use Kubernetes to deploy and manage the enabler's instances; however, this software is very complex and brings a heavy load into the system. Thus, researchers suggest Kubernetes works best for deploying complex applications; otherwise other orchestration tools such as Docker management are more appropriate [77].
- *Storage*: Persistent storage is provided by on-premise storage that contains all enabler information and the configuration specifications of the EF. This storage is a PostgreSQL database, and all data management operations performed over EF components are based on ORM (object–relation mapping), which reduces the complexity of accessing the database [78]. ORM functionality is available through the library of static data models.
- *Request Handler*: This module implements all necessary functionalities for translating application requests to the enablers that are made available by the EF. The internal submodules parse the requests and create a suitable communication channel for accessing the functionalities of the enablers and build the necessary response message based on the results of function invocation. This module uses the enabler technical details from the registry and has suitable proxies to invoke the services of the enabler as requested by the application. The current module allows integration of services that use NGSIV1, NGSIV2 and REST protocols because they are the interfaces required for vf-OS. Nonetheless, other protocols can be integrated into the Request Handler module to allow interoperability of services with other interfaces.

Taking into account that the EF submodule is responsible for making requests to the registered enablers, it can be configured to limit the number of requests to these enablers. This security measure is managed by the EF configuration GUI, which allows the administrator to define a security layer and control request throughput. This central module provides access to the registered enablers and also measures the time the enabler took to respond and any enabler errors or technical information. This information is sent to the enabler registry submodule to be associated with the enabler's registry information and become accessible to a monitoring service that will track possible service errors.

3.2. Enabler Data Structure

This subsection focuses on the granularity that an enabler component can have. Figure 4 represents the main classes of enablers and their hierarchical structure. Although this diagram was made for enablers, this object data model can be applied to any service/application deployed in an IoT environment. A software component possesses one or

multiple services (identified by their unique identifier), and it can have multiple versions and instances deployed and being used by the system.

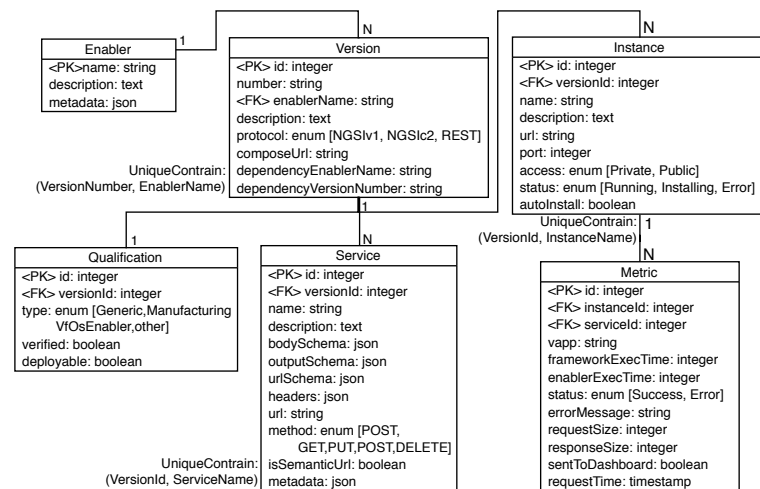


Figure 4. Enabler data structure class names and their relations. Attributes and parameters are not presented in order to simplify the diagram.

A description of the classes represented in Figure 4 is provided below:

- *Enabler*: Contains information that identifies one enabler. It contains a name that must be unique to avoid conflicts when the application is accessing the IoT software asset.
- *Version*: Each time a software component changes, it is a good practice to change its version number. This is due to the fact that some updates can impact service execution, such as different APIs or internal behaviour changes. To prevent interoperability problems when a software component is updated, the Enabler class is compatible with multiple versions.
- *Qualification*: Provides stability and trust for the associated version. Each time a new version is added, a specialized authority, e.g., security admin or software developer chief, must validate and confirm that this new version is working and can be used for future applications.
- *Service*: All available services from the particular version of a particular enabler's version are instances of this class. For each service, it is possible to specify the required inputs to execute the service, thus facilitating service execution. This class contains multiple fields (e.g., input headers, required URL parameters and input body schemas) to allow the specification of a wide range of services.
- *Instance*: Instances are software components that have all functionalities of the corresponding enabler. This feature allows replication of software components or having multiple versions of the same enabler within the same IoT system, as examples, to have software redundancy or software compatibility, respectively, without changing any software code.
- *Metric*: Metrics refers to quantitative measurements for quality engineering. This SOA characteristic is important when developing applications to guarantee Quality of Service, as it provides information on the availability of a service and also allows choice between identical services that have different response times.

Programmatically, the mentioned classes were developed using the Sequelize (<http://docs.sequelizejs.com>, (accessed on 28 July 2022)) library—an ORM framework that allows developers to define static models for all information that is stored in a database. This high-level abstraction enables management of objects, inheritances and properties related to the service execution without using SQL queries. This database management system is used by the EF only to abstract DB-specific implementation details and facilitate data access.

3.3. Instantiation of Enablers

Instance creation and installation are performed by establishment of a Docker container from the enabler's image, as described in [79]. To run an enabler on a personal computer or public server, it is necessary to have access to its virtual instance. The Docker solution allows instantiation of enablers, following an approach similar to the one for microservices described in [80] presenting an alternative to Virtual Machines. Docker is a platform that lets users run applications in an isolated environment, i.e., it is possible to run multiple containers at the same time on the same host. Using this approach, it is possible to package any component into isolated containers with required code, runtime environment, system tools, system libraries and settings all bundled together. This strategy can provide efficient, lightweight, self-contained software packages and guarantees that software always runs the same way regardless of where it is deployed. A Docker-based approach ensures enabler instantiation, granting that all enablers can be installed, thus contributing to the integration of the overall system.

To perform the functions mentioned above, the EF possesses specific Docker services, allowing it to deploy Docker instances. Taking into consideration that EF is used as a bridge between external services (enablers) and applications, deep analysis (e.g., software tests and documentation analysis) of the enabler's communication protocol is recommended in order to guarantee interoperability between components. Figure 5 illustrates the modular approach for Dockerisation in IoT using software-based solutions.

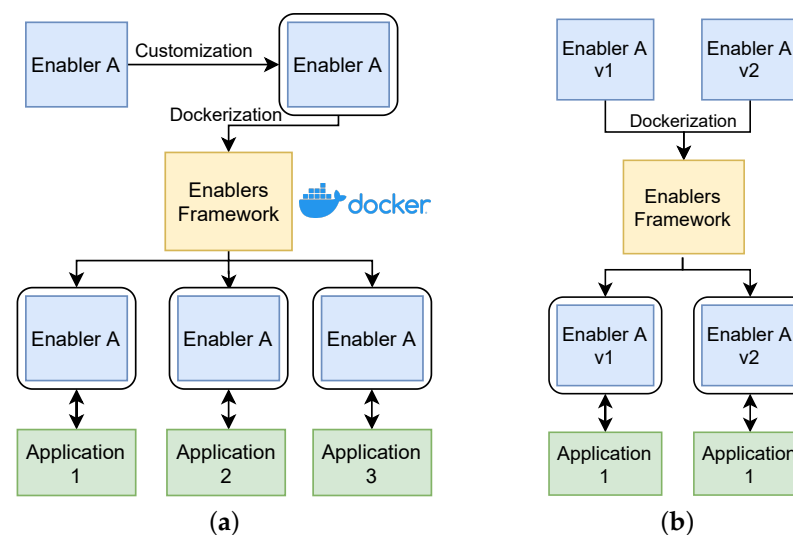


Figure 5. Dockerisation within EF: (a) enabler Dockerisation and replication on an IoT environment; (b) an application can choose the version that is compatible.

Figure 5a represents the capability for replication of enablers and their services. First, it is necessary to preconfigure the enabler and then upload it into a Docker Hub. Afterwards, the EF will download and install that preconfigured enabler. In the end, each application has the possibility to instantiate its enabler. This contributes not only to customization of their inner content, but also improves throughput by having available service replicas of the same enabler. Moreover, from a security perspective, major IoT environments use external services to compute or store sensor data. In this regard, there is a need that critical components or functions are replicated and have high availability to increase reliability of the system by ensuring components are always available to perform their actions.

Figure 5b shows how an application can use other versions of the same enabler. By having accessible and integrated old and new enabler versions, the developer of an application can choose and test different versions of the same component. In this fashion, there is the opportunity to validate that a new version of an enabler does not negatively affect the application (e.g., unstable versions and revoked functionalities). In software,

new versions might not only bring the new and upgraded features related to the core functionality, but can also include security improvements.

3.4. Enhancing Security with Enablers Framework

In the proposed scenario, two different components responsible for security are used, as mentioned in the next subsection. Security has always been a matter of extreme relevance as a result of the increased use of intelligent systems dealing with sensitive data by businesses in various industries worldwide.

An alternative type of privacy-enhancing technology that ensures data integrity is Transport Layer Security (TLS). This type of security protocol provides enhanced privacy and data integrity between two networks. The use of such a protocol improves the privacy of users and the confidentiality of sensitive data exchanged between intelligent IoT systems [81]. The proposed data-middleware enablers for security, presented in the next subsection, provide this security feature by enabling TLS on the message broker. Thus, IoT technologies will be viewed by enterprises as a solution with greater integrity while maintaining added value since business operations would be optimized without the risk of compromising sensitive data.

However, there are some implications related to the utilization of existing security- and privacy-enhancing enablers by EF. The most important challenge is the customization and adoption of security-enhancing components by different applications of the system that have their specific interfaces to access security functionalities. One possibility to overcome this implication is to integrate the EF with the Zero-Defects Manufacturing Platform (ZDMP) [82] so that the EF can use the security mechanisms implemented within ZDMP. ZDMP is a European manufacturing project that focuses on using different technological concepts, including IoT, to improve the quality assessment process in the manufacturing domain. Moreover, it provides a platform on which different services are available, including ones addressing security and privacy issues. All security functionalities are aggregated by the Security Command Centre (SCC) and can be managed/administrated from within the SCC by authorized users. The following core security issues are addressed by ZDMP security components:

- *Secure Communication*: This issue is addressed by withdrawal, renewal and issuing of digital certificates to enable secure data exchange between system components and external resources. All components within the system environment should possess a certificate for secure communication with other components. Thus, only authentic users and assets are allowed to exchange data and to be sure that data are genuine. Certificates are needed both to encrypt and authenticate communication channels and transmitted data.
- *Secure Installation*: Considering the potential risks third-party applications present to an IoT system, there is a necessity to use mechanisms to minimize these security risks, both for data and infrastructure. In this regard, the security measures provided are: (i) signature verification, (ii) behaviour monitoring during application run-time to detect any deviations or anomalies, and (iii) security policy management to limit accessible data to the minimum required by the service/application.
- *Authorization and authentication*: Authentication is the process of validation of user or asset identity, while authorization starts afterward to verify if the authenticated person/asset is allowed to access the requested resources. Authentication and authorization management is available from within the SCC, while the system assets can acquire functionality using the REST API. The authorization process relies heavily on OAuth 2.0 protocol combined with Role-Based Access Control and Attribute-Based Access Control.

ZDMP follows the recommendations described within the General Data Protection Regulation (GDPR), to ensure privacy perseverance. Based on this, ZDMP acts in accordance with the following principles:

- “*Privacy by design*”: A key principle that is applied to all activities involving the processing of personal data. This presumes the introduction of data protection from the very beginning the design of the product, service and application. This principle is applied for all the cases where ZDMP or its partners have the role of data controller, i.e., “the natural or legal person, public authority, agency, or another body which, alone or jointly with others, determines the purposes and means of the processing of personal data” [83].
- “*Privacy by default*”: This is applied to all activities involving the processing of personal data. This means that all the partners involved in ZDMP implement and ensure collection of only the necessary personal data for all the activities they are involved as controllers in. Thus, each process involving personal data processing is checked for the possibility of decreasing the personal data required to reach the same or a comparable result.

4. Methods for Prototypical Implementation of an IoT Scenario

The availability of data is a key element for IoT applications [13]. In IoT, data mainly refer to data from sensors or any network infrastructure relevant for the applications, bringing opportunities for devices, services and users to share and exchange information over the network. As mentioned before, to ease this interoperability between IoT components, the reutilization of existing services allow integration of multiple software modules and their use on different domains. Thus, this section presents an industrial scenario to demonstrate the applicability of the proposed EF for IOT applications development according to such characteristics as multiple software module integration and reutilization. Additionally, this scenario may be applied to any other IoT domain, such as agriculture, domotics or healthcare.

4.1. Industrial Scenario

The proposed scenario is based on the architecture presented in Section 3.1, making use of the data from sensors to manage a supply chain within an industrial environment, represented in Figure 6. Taking advantage of sensors placed on different locations of the factory, the management of products can be significantly improved. A particular example focuses on scheduling when the final product is ready to be delivered to the customer. This figure contains real-time data applications that are responsible for executing a set of services for specific use-cases, presented in Section 4.3. The collected data are also used by internal components to trigger alarms when a specific threshold is crossed. The authors also take into consideration the security principles that are important and necessary for an industrial scenario in order to prove the applicability of the proposed framework.

As depicted in Figure 6, the industrial applications, represented by green tabs, use different enablers, represented by blue tabs, and the external module is represented by a grey tab. The arrows interconnect the enablers/components and industrial applications. The colour of these arrows, namely yellow and purple, corresponds to the specific application, Supply Analytic and Product Alarm, respectively. The black arrows stand for the allowed interactions between the software modules of the proposed scenario.

Integration of these components is a challenging task because each dependent module has a different communication protocol and a different API. To overcome this interoperability problem, all applications use the EF as a bridge between industrial applications and corresponding enablers/components.

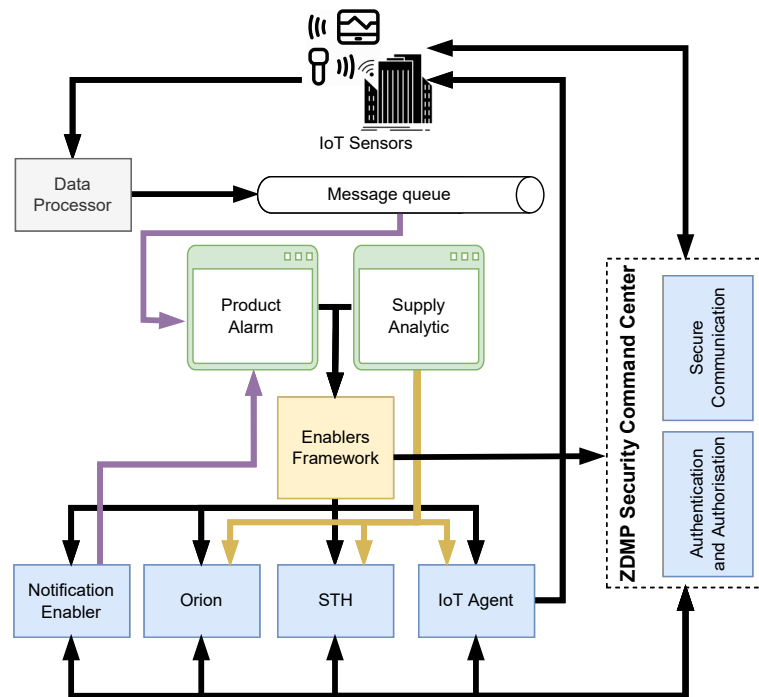


Figure 6. Industrial scenario using enablers to create an IoT environment. Both ZDMP and enablers can be accessed by one software module, EF, that provides the necessary services for the IoT system.

4.2. Components

The components that are involved in the proposed scenario were selected based on their relevance for the vf-OS project and security features adopted from ZDMP. These components are represented in Figure 6 by blue and grey tabs, and their descriptions are listed below.

- *Message Queue and Data Processor:* Responsible for communication and data-flow management between the connected applications and IoT sensors. They are composed of a RabbitMQ (www.rabbitmq.com, (accessed on 28 July 2022)) message broker and Data Processor grey box from Figure 6. The message queue receives all data from the enablers and redirects them to the correct receivers through the Data Processor component. The modules that use this messaging functionality have access to the software library, which provides functionalities for communication with the broker using the AMQP (www.amqp.org, (accessed on 28 July 2022)) protocol. This protocol is a middleware messaging standard, and it can be applied following publish/subscribe or point-to-point communication patterns.
- *Enablers Framework:* Middleware component responsible for the integration of enablers so that different enablers can be uniformly accessed and utilized.
- *Orion Context Broker:* GE that allows management of context information. This enabler operates with the notions of entity (e.g., house, room, or car) and a set of attributes intrinsic to the entity with the capability to query, update and subscribe established entities.
- *Short-Time Historic (STH):* A GE that manages (stores and retrieves) historical and aggregated time-series data. It reflects the changes in the context data, for instance, updates of attribute values.
- *IoT Agent:* GE used to manage connections of edge devices, such as sensors, providing corresponding APIs. The sensors are connected with a driver module, a component that is responsible for interpreting the different sensor protocols in order to connect to the message broker.
- *Notification Enabler:* Specific enabler component that provides a notification functionality based on predefined rules, for example, if the room temperature is higher than a

specific value. If these rules are triggered, the user will be notified through an HTTP call or by email [84].

- *ZDMP Security Command Center*: Two specific enablers that were developed for ZDMP that address the following security concerns:
 - *Authentication and Authorization*: This enabler is used to provide authorization and authentication for users and assets (other enablers and edge devices). The enabler can: (i) store authentication credentials, (ii) issue, after successful authentication, and store access tokens, (iii) store logs of all authentication attempts, (iv) manage the access policy for the registered assets, and (v) detect suspicious activity while monitoring communication among users and assets.
 - *Secure Communication*: This enabler addresses the issuing and revoking of digital certificates for secure communication among users and assets, both internal and external. The enabler includes a Certification Authority and a Registration Authority. The secure communication enabler is tightly coupled with the authentication and authorisation enabler, so that authenticated users or assets can get corresponding digital certificates to collaborate with each other.

4.3. Applications

The applications involved in this scenario are represented by a green application frame in Figure 6, and their descriptions are provided below:

- *SupplyAnalytic*: All sensors from the environment generate large amounts of data, which are useless unless analysed and interpreted. This application uses the data gathered from the sensors deployed in the physical environment to know the past and current supply chain context. With this information, it is possible to determine how the product evolves over time and, in case of success or failure, repeat or avoid past production methodologies accordingly. The enabler responsible for context information management receives data from sensors to simplify interpretation. Moreover, another enabler stores historic data. When actuators are involved, the IoT agent component is used to automatically trigger the control output.
- *ProductAlarm*: The real-time size of each product from a factory production line is used as an input for this application. Data preprocessing (filtering, compression and aggregation) is then performed to examine the status of the production line. According to the factory's current context values, if the product has not reached a predefined size threshold, an email is sent to the responsible parties. This product-monitor application can be used to detect and prevent faulty products in a production line when machinery calibration is required. Many factory machines require calibration when specific problems occur, and if it is not done quickly, the output might be incorrect. In a factory, the late detection of these problems can lead to severe economic losses.

5. Results

The proposed architecture from Section 3.1 was validated in an industrial scenario by resorting to already-developed modules, i.e., GEs and specific enablers, with the objectives of accelerating the development of new applications, reducing the cost of creating applications from scratch, and seamlessly integrating factory devices and factory data analytics by utilizing EF functionalities. The two applications and enabler instances presented in this section were deployed on the same machine and using, when necessary, the same enabler instance. This is due to the fact that the presented scenario covers the same IoT system, so the data are freely shared among all applications. Furthermore, all enablers, their versions and instances were registered on the EF and were available to internal and external users. This registration allows central and common access to the enabler services by the developer of the application.

As developers start to integrate the inner components of their IoT applications, it is helpful to know how the components behave, e.g., their response time or their availability and error handling. Figure 7 shows the Enabler Registry user interface from EF with all GEs

described in Section 4.2 that were involved in the proposed scenario. This main page allows EF configuration and the registration of new enablers to be used within the IoT system.

Name	Actions	Versions
Orion	Edit Delete	Versions
Notification Enabler	Edit Delete	Versions
STH	Edit Delete	Versions
Authentication and Authorization	Edit Delete	Versions

Figure 7. Enabler Registry management of enablers and their services.

One of the purposes of EF is to provide awareness to the developer during the run-time of the enabler through different measurements, as illustrated in Figure 8. These metrics indicate the execution time of the enabler, explore if any errors occurred, and provide data size and run-time duration. As such, these measurements can indicate the successfulness of enabler integration and usage.

SN	vApp	Framework ExecTime	Enabler ExecTime	Status	Error Message	Request Size
1	test8	74550	58634	Success		308
2	test8	142881	132398	Success		176
3	test8	29288	9471	Success		128
4	test9	21891	5283	Success		128
5	test7	136249	97627	Error	Error: getaddrinfo ENOTFOUND notification5133	170
6	test9	30255	12414	Success		662
7	test9	38955	24122	Success		216
8	test9	28372	7051	Error	404 - * {\"success\":true,\"d oes not exist\"}*	176

Figure 8. Measurements of enabler services through quantifiable and countable characteristics.

In the next step, configuration of all enabler instances and security components is accomplished. For example, the Notification Enabler component is configured, meaning the product threshold is set so that every threshold violation triggers an alarm, e.g., if the size of a product from a production line is not higher than a predefined measurement. The application scenario presented in Section 4.3 addresses the steps represented in Figures 9 and 10. As such, Figure 9 shows an output example of an Orion enabler based on a sensor value. The integration of this enabler allows the creation of applications that use contextual information to describe and act according to the context defined by the user, the system equipment and the surrounding environment. Figure 10 is an example of

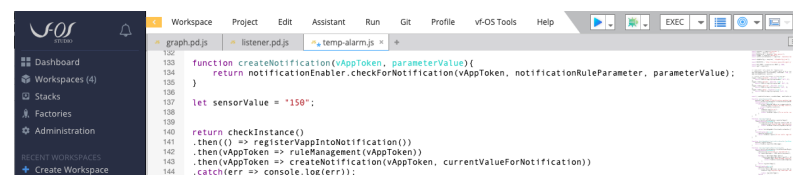
how the application integrates the information from the IoT (sensor attribute) and sends a notification by means of the Notification Enabler to the target user. Here, the application is registered at the Notification Enabler, and run-time configurations are made (e.g., the factory supervisor email is defined). Then, the context of the factory is acquired, and these values are submitted to the Notification Enabler to trigger an alarm if the previous values exceed the threshold.

```

{
  "contextResponses": [
    {
      "contextElement": {
        "type": "Sensor",
        "isPattern": "false",
        "id": "Sensor1",
        "attributes": [
          {
            "name": "value",
            "type": "int",
            "value": "37"
          }
        ]
      },
      "statusCode": {
        "code": "200",
        "reasonPhrase": "OK"
      }
    }
  ]
}

```

Figure 9. Orion sensor information after receiving data from IoT agent.



The screenshot shows a development environment with a sidebar on the left containing navigation options like Dashboard, Workspaces (4), Stacks, Factories, and Administration. The main area displays a code editor with the following JavaScript code:

```

133 function createNotification(vAppToken, parameterValue){
134   return notificationEnabler.checkForNotification(vAppToken, notificationRuleParameter, parameterValue);
135 }
136
137 let sensorValue = "150";
138
139
140 return checkInstance()
141 .then(() => registerVappIntokNotification())
142 .then(vAppToken => ruleManagement(vAppToken))
143 .then(vAppToken => createNotification(vAppToken, currentValueForNotification))
144 .catch(err => console.log(err));

```

Figure 10. Example of the execution of a service using EF; in particular, sending a notification based on a sensor value.

6. Discussion

The IT infrastructure of industrial and enterprise environments has always been evolving. Since industrial machinery commonly used in factories needs to be enhanced over time to add functionalities and to improve those already available, their hardware/software must also be adjusted to these changes, which can be seamlessly made through changing/adding new components to the process workflow [85]. Yelamarthi et al. [86] claim that most of the existing IoT architectures are not prepared to be used in various application domains. To overcome this adaptation problem, the authors propose a modular IoT architecture that can be configured for different domains, such as Smart Homes or Smart Cities, industrial environment control for energy usage optimization and automation, agriculture for soil and temperature monitoring, healthcare for monitoring patient physiological state, or remote motion tracking [85].

In order to integrate IoT components within an industrial system, a layer that provides the necessary abstraction from the technical details of existing services is needed to reduce the complexity of the system components for the external developers. Thus, this abstraction layer serves as middleware to provide an intermediary layer between applications and services provided by the IoT platform. Development of such an abstraction layer requires the following implementation-level concerns to be considered [13]:

- *Programming abstraction:* The API for application developers must be easy and intuitive to use. When developing the middleware, programming paradigms and interface types must be well-defined. The abstraction level specifies how the user views the system (e.g., individual node/device level). The appropriate programming paradigm

simplifies the processes of modelling and programming, and the interface type defines the communication style and must be similar to the services to be more intuitive.

- *Service-based*: The process of adding new functions to IoT middleware has to be flexible and easy. To have the seamless integration of different components, the interfaces of those components must be standardized to facilitate interoperability [87]. It is also mandatory that each IoT module is uniquely defined, enabling discoverability within the object's network [17].

As the software industry is moving towards service-oriented integration, multiple services that use specific communication protocols co-exist; vivid examples are RESTful APIs or JSON RPC-based interfaces. For this reason, a service abstraction layer in the IoT architecture is rather important to allow interoperability between an IoT application and all available services, assuring correct information exchange. Thus, the presented framework can help in this interoperability process by offering such a common interface regardless of the system, i.e., REST API. This methodology allows new devices or software components that rely on different interfaces to still be able to communicate without affecting the consumers or additional integration efforts. As a result, any application that can communicate with the EF can also use the EF-registered services. Additionally, by introducing some metrics about the running services, such as execution time or error messages, administrators have improved knowledge about the services that are running on the system. As a result, the EF contributes to the service layer of SOA while allowing comparison, deployment and analysis of new services entering the existing system.

In our opinion, a big advantage of adopting IoT is the opportunity to make use of data-rich environments to produce new services for citizens, industry or even primary sectors (e.g., agriculture and aquaculture). The amount of data generated comes from a high number of internet-connected devices, which, subsequently, impose additional security and safety challenges to be managed. The potential security threats might cause significant consequences related to incorrect decisions in the product value chain, causing monetary losses.

The research presented in this document covers some aspects of IoT platforms, such as sensor data, security, integration and interoperability of software services containing different interfaces or communication protocols between different IoT components. The presented work can help other researchers and practitioners to understand possible ways to improve the interoperability of IoT solutions and create IoT applications that are useful for daily life. The presented framework is not static, and for this reason, if an IoT system requires integration of components with other communication interfaces that are not available in the EF, these components can be improved to allow the correct exchange of information.

The presented scenario is based on a multilayer generic and modular architecture. The scenario was developed and tested within the research performed in the scope of vf-OS and ZDMP H2020 research projects; we will formalize the definitions of specific enablers for ZDMP components, and they will be integrated with the EF in future work. As a final consideration and prospect for future work, it is considered that the applications presented in this document and those being developed are examples of the immense potential of IoT devices using a SOA-based framework. As demonstrated through this work, a service-oriented architecture provides high flexibility and integration functionalities, which are important for both academia and industry to conduct future analysis and investigations in the field of interoperability. Such an environment has multiple applications, such as those hereby presented and scalability to any other business activity where connectivity is essential and the IoT is a driver of innovation and real-world data collection and actuation. The present work aims to widen such a vision and foster new opportunities and new applications for IoT-based applications.

Author Contributions: J.G., F.L.-F. and J.S. conceived the presented idea. J.G., A.A.N. and D.G. developed the theoretical analysis. F.L.-F. and J.S. supervised the findings of this work. All authors provided critical feedback and helped shaping the research, discussed the results and contributed to the final manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by funds provided by the European Commission in the scope of FoF/H2020-723710 vf-OS, ICT/H2020-825631 ZDMP projects, and by the FCT—Fundação para a Ciência e a Tecnologia in the scope of UIDB/00066/2020 related to CTS—Centro de Tecnologia e Sistemas research unit.

Data Availability Statement: The framework presented in this work is available at <https://code.grisenergia.pt/vfos/enablers-framework> (accessed on 28 July 2022).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
GE	Generic Enablers
IIoT	Industrial IoT
IoT	Internet of Things
MSA	Microservice Architecture
ORM	Object-relation Mapping
SOA	Service-Oriented Architecture
SCC	Security Command Centre
TLS	Transport Layer Security
VPN	Virtual Private Networks
ZDMP	Zero-Defects Manufacturing Platform

References

- Blackstock, M.; Lea, R. IoT interoperability: A hub-based approach. In Proceedings of the 2014 International Conference on the Internet of Things (IOT), Cambridge, MA, USA, 6–8 October 2014. [CrossRef]
- Khan, F.; Tarimer, I.; Taekeun, W. Factor Model for Online Education during the COVID-19 Pandemic Using the IoT. *Processes* **2022**, *10*, 1419. [CrossRef]
- Li, S.; Xu, L.D.; Zhao, S. The internet of things: A survey. *Inf. Syst. Front.* **2015**, *17*, 243–259. [CrossRef]
- Saidu, C.; Usman, A.; Ogedebe, P. Internet of Things: Impact on Economy. *Br. J. Math. Comput. Sci.* **2015**, *7*, 241–251. [CrossRef]
- Manyika, J.; Dobbs, R.; Chui, M.; Bughin, J.; Bisson, P.; Woetzel, J. *The Internet of Things: Mapping the Value Beyond the Hype*; Technical Report; McKinsey Global Institute, McKinsey & Company: Hong Kong, China, 2015.
- Xie, J.; Chen, C. Supply chain and logistics optimization management for international trading enterprises using IoT-based economic logistics model. *Oper. Manag. Res.* **2022**. [CrossRef]
- Jiang, Z.; Chang, Y.; Liu, X. Design of software-defined gateway for industrial interconnection. *J. Ind. Inf. Integr.* **2020**, *18*, 100130. [CrossRef]
- Fortino, G.; Savaglio, C.; Palau, C.E.; de Puga, J.S.; Ganzha, M.; Paprzycki, M.; Montesinos, M.; Liotta, A.; Llop, M. Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach. In *Integration, Interconnection, and Interoperability of IoT Systems*; Internet of Things (Technology, Communications and Computing); Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 199–232. [CrossRef]
- Noura, M.; Atiquzzaman, M.; Gaedke, M. Interoperability in Internet of Things: Taxonomies and Open Challenges. *Mob. Netw. Appl.* **2019**, *24*, 796–809. [CrossRef]
- Costa, B.; Pires, P.F.; Delicato, F.C. Towards the adoption of OMG standards in the development of SOA-based IoT systems. *J. Syst. Softw.* **2020**, *169*, 110720. [CrossRef]
- Marks, E.A.; Bell, M. *Service-Oriented Architecture*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2012. [CrossRef]
- Kosanke, K. ISO Standards for Interoperability: A Comparison. In *Interoperability of Enterprise Software and Applications*; Konstantas, D., Bourrières, J.P., Léonard, M., Boudjlida, N., Eds.; Springer: London, UK, 2006; pp. 55–64. [CrossRef]
- Razzaque, M.A.; Milojevic-Jevric, M.; Palade, A.; Clarke, S. Middleware for Internet of Things: A Survey. *IEEE Internet Things J.* **2016**, *3*, 70–95. [CrossRef]
- Lu, Y. Industry 4.0: A survey on technologies, applications and open research issues. *J. Ind. Inf. Integr.* **2017**, *6*, 1–10. [CrossRef]
- Tran, K.P. Artificial Intelligence for Smart Manufacturing: Methods and Applications. *Sensors* **2021**, *21*, 5584. [CrossRef]
- Chen, Y. A Survey on Industrial Information Integration 2016–2019. *J. Ind. Integr. Manag.* **2020**, *5*, 33–163. [CrossRef]

17. Hejazi, H.; Rajab, H.; Cinkler, T.; Lengyel, L. Survey of platforms for massive IoT. In Proceedings of the 2018 IEEE International Conference on Future IoT Technologies (Future IoT), Eger, Hungary, 18–19 January 2018; pp. 1–8. [CrossRef]
18. Schneider, M.; Hippchen, B.; Abeck, S.; Jacoby, M.; Herzog, R. Enabling IoT Platform Interoperability Using a Systematic Development Approach by Example. In Proceedings of the 2018 Global Internet of Things Summit (GloTS), Bilbao, Spain, 4–7 June 2018; pp. 1–6. [CrossRef]
19. Sill, A. Standards at the Edge of the Cloud. *IEEE Cloud Comput.* **2017**, *4*, 63–67. [CrossRef]
20. Alsoubi, T.; Qin, Y.; Hill, R.; Al-Aqrabi, H. Distributed Intelligence in the Internet of Things: Challenges and Opportunities. *SN Comput. Sci.* **2021**, *2*, 277. [CrossRef]
21. López, E.J.; Jiménez, F.C.; Sandoval, G.L.; Estrella, F.J.O.; Monteón, M.A.M.; Muñoz, F.; Leyva, P.A.L. Technical Considerations for the Conformation of Specific Competences in Mechatronic Engineers in the Context of Industry 4.0 and 5.0. *Processes* **2022**, *10*, 1445. [CrossRef]
22. Tayur, V.M.; Suchithra, R. Review of interoperability approaches in application layer of Internet of Things. In Proceedings of the 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), Bengaluru, India, 21–23 February 2017; pp. 322–326. [CrossRef]
23. European Commission. The Future Internet Platform FIWARE. Available online: <https://ec.europa.eu/digital-single-market/en/future-internet-public-private-partnership> (accessed on 23 August 2018).
24. FIWARE. FIWARE Catalogue. Available online: <https://github.com/FIWARE/catalogue> (accessed on 25 November 2021).
25. Abid, M.A.; Afaqui, N.; Khan, M.A.; Akhtar, M.W.; Malik, A.W.; Munir, A.; Ahmad, J.; Shabir, B. Evolution towards Smart and Software-Defined Internet of Things. *AI* **2022**, *3*, 100–123. [CrossRef]
26. Ngu, A.H.H.; Gutierrez, M.; Metsis, V.; Nepal, S.; Sheng, M.Z. IoT Middleware: A Survey on Issues and Enabling technologies. *IEEE Internet Things J.* **2016**, *4*, 1–20. [CrossRef]
27. Zhang, J.; Ma, M.; Wang, P.; Dong Sun, X. Middleware for the Internet of Things: A survey on requirements, enabling technologies, and solutions. *J. Syst. Archit.* **2021**, *117*, 02098. [CrossRef]
28. Xia, F. QoS Challenges and Opportunities in Wireless Sensor/Actuator Networks. *Sensors* **2008**, *8*, 1099–1110. [CrossRef]
29. Kuehnel, K.; Au-Yong-Oliveira, M. The Development of an Information Technology Architecture for Automated, Agile and Versatile Companies with Ecological and Ethical Guidelines. *Informatics* **2022**, *9*, 37. [CrossRef]
30. Shaikh, A.; Reshan, M.S.A.; Sulaiman, A.; Alshahrani, H.; Asiri, Y. Secure Telemedicine System Design for COVID-19 Patients Treatment Using Service Oriented Architecture. *Sensors* **2022**, *22*, 952. [CrossRef]
31. Avila, K.; Sanmartin, P.; Jabba, D.; Jimeno, M. Applications Based on Service-Oriented Architecture (SOA) in the Field of Home Healthcare. *Sensors* **2017**, *17*, 1703. [CrossRef]
32. Chen, I.R.; Guo, J.; Bao, F. Trust Management for SOA-Based IoT and Its Application to Service Composition. *IEEE Trans. Serv. Comput.* **2016**, *9*, 482–495. [CrossRef]
33. Ochs, J.; Biermann, F.; Piotrowski, T.; Erkens, F.; Nießing, B.; Herbst, L.; König, N.; Schmitt, R.H. Fully Automated Cultivation of Adipose-Derived Stem Cells in the StemCellDiscovery—A Robotic Laboratory for Small-Scale, High-Throughput Cell Production Including Deep Learning-Based Confluence Estimation. *Processes* **2021**, *9*, 575. [CrossRef]
34. Kyösti, P.; Lindström, J. SOA-Based Platform Use in Development and Operation of Automation Solutions: Challenges, Opportunities, and Supporting Pillars towards Emerging Trends. *Appl. Sci.* **2022**, *12*, 1074. [CrossRef]
35. Niknejad, N.; Ismail, W.; Ghani, I.; Nazari, B.; Bahari, M.; Hussin, A.R.B.C. Understanding Service-Oriented Architecture (SOA): A systematic literature review and directions for further investigation. *Inf. Syst.* **2020**, *91*, 101491. [CrossRef]
36. Tiburski, R.T.; Amaral, L.A.; Matos, E.D.; Hessel, F. The importance of a standard security architecture for SOA-based iot middleware. *IEEE Commun. Mag.* **2015**, *53*, 20–26. [CrossRef]
37. Suljkanović, A.; Milosavljević, B.; Indić, V.; Dejanović, I. Developing Microservice-Based Applications Using the Silvera Domain-Specific Language. *Appl. Sci.* **2022**, *12*, 6679. [CrossRef]
38. Raj, V.; Sadam, R. Performance and complexity comparison of service oriented architecture and microservices architecture. *Int. J. Commun. Netw. Distrib. Syst.* **2021**, *27*, 100–117. [CrossRef]
39. Zhu, W.; Zhou, G.; Yen, I.L.; Bastani, F. A PT-SOA Model for CPS/IoT Services. In Proceedings of the 2015 IEEE International Conference on Web Services, New York, NY, USA, 27 June–2 July 2015. [CrossRef]
40. Uviase, O.; Kotonya, G. IoT Architectural Framework: Connection and Integration Framework for IoT Systems. *Electron. Proc. Theor. Comput. Sci.* **2018**, *264*, 1–17. [CrossRef]
41. W3C Semantic Sensor Network Incubator Group. Semantic Sensor Network Ontology. Available online: <https://www.w3.org/2005/Incubator/ssn/ssnx/ssn> (accessed on 25 November 2021).
42. W3C. Web of Things (WoT) Architecture. Available online: <https://www.w3.org/TR/wot-architecture/> (accessed on 25 November 2021).
43. ISO. ISO/IEC 30161:2020-Internet of Things (IoT) — Requirements of IoT Data Exchange Platform for Various IoT Services. Available online: <https://www.iso.org/standard/53281.html> (accessed on 25 November 2021).
44. ETSI. Smart Appliances and SAREF. Available online: <https://www.etsi.org/technologies/smart-appliances> (accessed on 25 November 2021).
45. Balaji, S.; Salih, A.; Al-Atroshi, C. Adaptability of SOA in IoT Services—An Empirical Survey. *Int. J. Comput. Appl.* **2018**, *182*, 25–28. [CrossRef]

46. Bandyopadhyay, S.; Sengupta, M.; Maiti, S.; Dutta, S. Role Of Middleware For Internet Of Things: A Study. *Int. J. Comput. Sci. Eng. Surv.* **2011**, *2*, 94–105. [CrossRef]
47. Alfalouji, Q.; Schranz, T.; Kümpel, A.; Schraven, M.; Storek, T.; Gross, S.; Monti, A.; Müller, D.; Schweiger, G. IoT Middleware Platforms for Smart Energy Systems: An Empirical Expert Survey. *Buildings* **2022**, *12*, 526. [CrossRef]
48. Palade, A.; Cabrera, C.; Li, F.; White, G.; Razzaque, M.A.; Clarke, S. Middleware for internet of things: An evaluation in a small-scale IoT environment. *J. Reliab. Intell. Environ.* **2018**, *4*, 3–23. [CrossRef]
49. da Cruz, M.A.A.; Rodrigues, J.J.P.C.; Sangaiah, A.K.; Al-Muhtadi, J.; Korotaev, V. Performance evaluation of IoT middleware. *J. Netw. Comput. Appl.* **2018**, *109*, 53–65. [CrossRef]
50. FIWARE. Orion Context Broker. Available online: <https://fiware-orion.readthedocs.io/> (accessed on 14 August 2022).
51. Martigne, P. Overview of ETSI machine-to-machine and oneM2M architectures. In *Machine-to-Machine (M2M) Communications*; Elsevier: Amsterdam, The Netherlands, 2015; pp. 27–46. [CrossRef]
52. Pereira, C.; Pinto, A.; Aguiar, A.; Rocha, P.; Santiago, F.; Sousa, J. IoT interoperability for actuating applications through standardised M2M communications. In Proceedings of the 2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Coimbra, Portugal, 21–24 June 2016; pp. 1–6. [CrossRef]
53. Cardoso, J.; Pereira, C.; Aguiar, A.; Morla, R. Benchmarking IoT middleware platforms. In Proceedings of the 2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), Macau, China, 12–15 June 2017; pp. 1–7. [CrossRef]
54. Arndt, M.; Koss, J. *ETSI M2M Horizontal Platform Strategy*; Technical Report; ETSI: Sophia Antipolis, France, 2014.
55. Lee, E.A. Accessors: What Are Accessors? Available online: <https://ptolemy.berkeley.edu/accessors> (accessed on 28 November 2021).
56. Accessors. Available online: <https://wiki.eecs.berkeley.edu/accessors/Version1/AccessorSpecification> (accessed on 14 August 2022).
57. Gama, K.; Touseau, L.; Donsez, D. Combining heterogeneous service technologies for building an Internet of Things middleware. *Comput. Commun.* **2012**, *35*, 405–417. [CrossRef]
58. Mynzhasova, A.; Radojicic, C.; Heinz, C.; Kolsch, J.; Grimm, C.; Rico, J.; Dickerson, K.; Garcia-Castro, R.; Oravec, V. Drivers, standards and platforms for the IoT: Towards a digital VICINITY. In Proceedings of the 2017 Intelligent Systems Conference (IntelliSys), London, UK, 7–8 September 2017; pp. 170–176. [CrossRef]
59. Lyaskov, M.; Spasov, G.; Petrova, G. A practical implementation of smart home energy data storage and control application based on cloud services. In Proceedings of the 2017 XXVI International Scientific Conference Electronics (ET), Sozopol, Bulgaria, 13–15 September 2017; pp. 1–4. [CrossRef]
60. DeviceHive. Three Steps To IoT. Available online: <https://docs.devicehive.com/docs> (accessed on 14 August 2022).
61. Protic, A.; Jin, Z.; Marian, R.; Abd, K.; Campbell, D.; Chahl, J. Implementation of a Bi-Directional Digital Twin for Industry 4 Labs in Academia: A Solution Based on OPC UA. In Proceedings of the 2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Singapore, 14–17 December 2020; pp. 979–983. [CrossRef]
62. Automation, I. Solving SCADA Pain Points: Why SCADA Is Broken & How Ignition Can Fix It. Available online: https://inductiveautomation.com/static/pdf/Solving_SCADA_Pain_Points_04-17-2018.pdf (accessed on 28 November 2021).
63. Automation, I. Diagnostics-Metrics Dashboard. Available online: <https://docs.inductiveautomation.com/display/DOC81/Diagnostics++Metrics+Dashboard> (accessed on 14 August 2022).
64. Balaji, S.; Nathani, K.; Santhakumar, R. IoT Technology, Applications and Challenges: A Contemporary Survey. *Wirel. Pers. Commun.* **2019**, *108*, 363–388. [CrossRef]
65. Menzel, L.M. Investigating the Adoption and Management of Metrics in Large-Scale Agile Software Development at a German IT-Provider. Master's Thesis, Technische Universität München, Munich, Germany, 2021.
66. Osório, A.L.; Camarinha-Matos, L.M.; Afsarmanesh, H.; Belloum, A. On Reliable Collaborative Mobility Services. In *IFIP Advances in Information and Communication Technology*; Camarinha-Matos, L.M., Afsarmanesh, H., Rezgui, Y., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2018; pp. 297–311. [CrossRef]
67. Razzaq, M.A.; Habib, S.; Ali, M.; Ullah, S. Security Issues in the Internet of Things (IoT): A Comprehensive Study. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 383–388. [CrossRef]
68. Shaikh, E.; Mohiuddin, I.; Manzoor, A. Internet of Things (IoT): Security and Privacy Threats. In Proceedings of the 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 1–3 May 2019; pp. 1–6. [CrossRef]
69. Gritzalis, D.A.; Pantziou, G.; Román-Castro, R. Sensors Cybersecurity. *Sensors* **2021**, *21*, 1762. [CrossRef]
70. Saif, I.; Peasley, S.; Perinkolam, A. *Safeguarding the Internet of Things*; Technical Report 17; Deloitte Review: Chiyoda, Tokyo, 2015.
71. Mai, J.; Du, J. BGP performance analysis for large scale VPN. In Proceedings of the 2013 IEEE Third International Conference on Information Science and Technology (ICIST), Yangzhou, China, 23–25 March 2013. [CrossRef]
72. Lagsaiar, L.; Shahrour, I.; Aljer, A.; Soulhi, A. Modular Software Architecture for Local Smart Building Servers. *Sensors* **2021**, *21*, 5810. [CrossRef]
73. Corista, P.; Ferreira, D.; Gao, J.; Sarraipa, J.; Goncalves, R.J. An IoT Agriculture System Using FIWARE. In Proceedings of the 2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), Stuttgart, Germany, 17–20 June 2018; pp. 1–6. [CrossRef]
74. FIWARE. Ngsijs Documentation. Available online: <https://conwetlab.github.io/ngsijs/stable/index.html> (accessed on 9 March 2019).

75. Information Catalyst for Entepresi LTD. Virtual Factory Open Operating System-CORDIS. Available online: <https://cordis.europa.eu/project/id/723710> (accessed on 7 October 2021).
76. FIWARE. FIWARE Contribution Requirements. Available online: <https://fiware-requirements.readthedocs.io/en/latest/> (accessed on 14 September 2021).
77. Chan, C. Autoscaling Cloud-Native Applications using Custom Controller of Kubernetes. Master's Thesis, National College of Ireland, Dublin, UK, 2021.
78. Chen, T.; Shang, W.; Yang, J.; Hassan, A.E.; Godfrey, M.W.; Nasser, M.; Flora, P. An Empirical Study on the Practice of Maintaining Object-Relational Mapping Code in Java Systems. In Proceedings of the 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), Austin, TX, USA, 14–15 May 2016; IEEE: New York, NY, USA; Los Alamitos, CA, USA, 2016; pp. 165–176.
79. Merkel, D. Docker: Lightweight Linux containers for consistent development and deployment. *Linux J.* **2014**, *239*, 2.
80. Stubbs, J.; Moreira, W.; Dooley, R. Distributed Systems of Microservices Using Docker and Serfnode. In Proceedings of the 2015 7th International Workshop on Science Gateways, Budapest, Hungary, 3–5 June 2015; pp. 34–39. [[CrossRef](#)]
81. Weber, R.H. Internet of Things-New security and privacy challenges. *Comput. Law Secur. Rev.* **2010**, *26*, 23–30. [[CrossRef](#)]
82. Nazarenko, A.A.; Lopes, C.; Ferreira, J.; Usher, P.; Sarraipa, J. ZDMP Core Services and Middleware. In Proceedings of the Workshops of I-ESA 2020, Tarbes, France, 17–19 November 2020.
83. ZDMP Consortium. WP2 Business Challenge: Vision, Market, Use Cases, and Interlinking-D2.5a: Regulation and Trustworthy System-Vs: 1.0.1; Technical Report; 2020. Available online: https://www.zdmp.eu/_files/ugd/f83381_2bc34c64f6fb4e708d8a507e94f86de7.pdf (accessed on 28 July 2022).
84. vf-OS Consortium. WP3: Virtual Factory System Kernel D3.1c: WP3 Umbrella Deliverable-Vs: 1.0; Technical Report; 2018. Available online: <https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5ce27dd89&appId=PPGMS> (accessed on 28 July 2022)
85. Gao, J.; Sarraipa, J.; Jardim-Gonçalves, R. Open Modular Components in the Industry Using vf-OS Components. In *DoCEIS 2019: Technological Innovation for Industry and Service Systems*; IFIP Advances in Information and Communication Technology; Springer: Cham, Switzerland, 2019; Volume 553, pp. 238–246. [[CrossRef](#)]
86. Yelamarthi, K.; Aman, M.S.; Abdelgawad, A. An Application-Driven Modular IoT Architecture. *Wirel. Commun. Mob. Comput.* **2017**, *2017*, 1–16. [[CrossRef](#)] [[PubMed](#)]
87. Baheti, R.; Gill, H. Cyber-Physical Systems. In *The Impact of Control Technology*; IEEE Control Systems Society: New York, NY, USA, 2011; Chapter Cross-Cutting Research Directions; pp. 161–166.