applied sciences

MDPI

Article

Context-Aware System for Information Flow Management in Factories of the Future

Pedro Monteiro [1], Rodrigo Pereira [1], Ricardo Nunes [1], Arsénio Reis [1,2] and Tiago Pinto [1,2,*]

[1] School of Science and Technology, Universidade de Trás-os-Montes e Alto Douro, 5000-801 Vila Real, Portugal; rrnunes@utad.pt (R.N.); ars@utad.pt (A.R.)
[2] INESC-TEC, Vila Real Pole, 5000-801 Vila Real, Portugal
* Correspondence: tiagopinto@utad.pt

Abstract: The trends of the 21st century are challenging the traditional production process due to the reduction in the life cycle of products and the demand for more complex products in greater quantities. Industry 4.0 (I4.0) was introduced in 2011 and it is recognized as the fourth industrial revolution, with the aim of improving manufacturing processes and increasing the competitiveness of industry. I4.0 uses technological concepts such as Cyber-Physical Systems, Internet of Things and Cloud Computing to create services, reduce costs and increase productivity. In addition, concepts such as Smart Factories are emerging, which use context awareness to assist people and optimize tasks based on data from the physical and virtual world. This article explores and applies the capabilities of context-aware applications in industry, with a focus on production lines. In specific, this paper proposes a context-aware application based on a microservices approach, intended for integration into a context-aware information system, with specific application in the area of manufacturing. The manuscript presents a detailed architecture for structuring the application, explaining components, functions and contributions. The discussion covers development technologies, integration and communication between the application and other services, as well as experimental findings, which demonstrate the applicability and advantages of the proposed solution.

Keywords: context; context awareness; context-aware applications; Industry 4.0; Internet of Things; microservices

Citation: Monteiro, P.; Pereira, R.; Nunes, R.; Reis, A.; Pinto, T. Context-Aware System for Information Flow Management in Factories of the Future. *Appl. Sci.* **2024**, *14*, 3907. https://doi.org/10.3390/app14093907

Academic Editor: Redha Taiar

Received: 22 March 2024
Revised: 26 April 2024
Accepted: 30 April 2024
Published: 3 May 2024

1. Introduction

Many organizations are currently integrating innovative technologies into their production systems to increase efficiency and improve product quality. The concept of Industry 4.0 (I4.0) is becoming widely recognized by organizations due to the benefits it brings to production and manufacturing processes [1]. The implementation of I4.0 in manufacturing companies is driven by technology. Described as the fourth Industrial Revolution (IR), I4.0 represents the current trend towards automation in the manufacturing industry, especially encompassing enabling technologies [2].

Throughout history, we have witnessed major transformations in industry, marked by the so called IRs. These revolutions, characterized by technological advances and innovations in industry over certain periods of time, have a huge impact on that sector and bring benefits to society [3].

Industry has already gone through four of these revolutions and is currently in the midst of the Fourth IR, known as I4.0. Compared to its predecessors, this IR is progressing at a significantly high speed, resulting in the disruption of almost all industries worldwide and leading to the transformation of entire production systems [4].

This revolution aims to strengthen the competitiveness of the manufacturing industry by targeting improvements in industrial processes [5]. The central idea of I4.0 is centered on technology and, above all, on the production area, but it is possible to apply this concept to any type of company [6].

I4.0 enables new technological ideas such as Cyber-Physical Systems (CPS), Internet of Things (IoT) and Cloud Computing to develop new services [6]. The adoption of I4.0 technologies enables more flexible production on a large scale, resulting in reduced lead times and increased efficiency [7]. Individually or collectively, the technologies incorporated into I4.0 promise to reduce costs and improve quality in manufacturing environments [8]. In addition, concepts such as Smart Factories are emerging, which use context awareness to assist people and optimize tasks based on data from the physical and virtual world [5].

Smart factories are made possible by advances in sensor technologies and wireless communications to automate the collection and distribution of information from physical events [9]. In these factories, technical assistance systems augment human skills, reducing errors and facilitating correct decisions [10]. Smart manufacturing uses advanced technologies for flexible, intelligent and reconfigurable processes, optimizing production and improving the product lifecycle [11]. It is characterized by the autonomy and decentralization of the agents involved, being highly integrated through the use of network technologies and manufacturing data, and is attracting great interest from companies [12].

Twenty-first century trends challenge production due to shorter life cycles, demand for complex products and unsustainable resource use practices [13]. Companies face significant challenges in the management and complexity of their production lines, where it is essential to maintain good performance [14].

In this situation, context-aware applications can play a key role in helping companies facing these challenges. They have the ability to understand the context of the factory in real time, processing it and generating relevant information for employees [15]. Context-aware applications are being implemented in various areas of the commercial sector, including manufacturing and smart factories, to assist Industry 4.0. Based on the context of the factories, these applications provide relevant information to workers through recommendations, assisting them with their tasks and helping to improve the management of the production process [16]. In this way, they enable workers to make decisions based on more comprehensive information, rather than relying on a fragmented view of the factory and limited specialist knowledge.

This article aims to explore and apply the capabilities of context-aware applications in industry, with a focus on production lines. This is accomplished by proposing and describing the development of a context-aware application, based on a microservices' architecture, intended for integration into a context-aware information system, with specific application in the area of manufacturing. The solution description includes the description of a detailed architecture for structuring the application, explaining components, functions and contributions. The discussion covers development technologies, integration and communication between the application and other services, as well as experimental findings.

The article is organized as follows: Section 2 presents the theoretical framework, establishing the theoretical basis of the key themes. Section 3 discusses the application development process, detailing the various choices made for its realization. In Section 4, the tests carried out on this application are discussed and some case studies are presented which provide a more detailed and real insight into the contribution and use of the application developed. Finally, Section 5 presents the most relevant conclusions of the work, as well as suggestions for future work.

2. Theoretical Framework

This section aims to establish a theoretical and conceptual foundation for various topics addressed throughout this article. Here, relevant literature on these topics is reviewed and synthesized, thus assisting in contextualizing the topics addressed later in this article.

2.1. Industry 4.0

The term "Industry 4.0" refers to the fourth IR and became publicly known in Germany in 2011 [5]. I4.0 promotes the idea of the start of a new IR, driven by the advance and

convergence of various technologies that make it possible to connect the physical and digital worlds in real time [8].

One of the foundations of I4.0 lies in the introduction of interconnected intelligent systems that enable self-sustainable production, in which people, machines, equipment and products communicate with each other [13]. I4.0 is made possible by Internet technologies to create intelligent products, intelligent production and intelligent services, since, from a communications point of view, I4.0 technologies rely heavily on the mobile internet [17]. This revolution aims to boost the competitiveness of the manufacturing sector by improving industrial procedures associated with production, engineering, the application of materials, supply chain management and the product life cycle [5].

I4.0 incorporates various innovative concepts, such as smart factories (production is equipped with intelligent components), Cyber-Physical Systems (CPS, resulting from the combination of the physical and digital worlds, allowing real-time integration of information), new systems in the development of products and services, adaptation to human needs, among others [18].

I4.0 is driven by four fundamental principles: [5]

- Interconnection—involves linking machines, devices, sensors and people;
- Information transparency—interconnects the physical and virtual worlds, providing contextual data in real time for appropriate decision-making;
- Decentralization of decisions—combines local and global information to increase productivity at different levels, this is possible thanks to systems that monitor and control the physical world autonomously;
- Technical assistance—uses mobile devices and robotic technology to assist in decision-making and problem-solving.

The implementation of I4.0 faces a number of challenges in different areas, such as science, technology, the economy, society and politics. It is essential to train workers to use the new technologies. In addition, business issues related to innovation, technological components and digital transformation represent major obstacles. Another major difficulty lies in data analysis and management [19].

2.2. Context-Aware Applications

Context-aware computing was first addressed by Schilit and Theimer in 1994 and since then, there have been numerous attempts to define this concept [20].

Context is any information that characterizes the situation of an entity, which can be a person, place or object considered relevant to the interaction between the user and the application. Context-aware applications use context to provide relevant information and/or services to the user, where relevance depends on the user's task [21].

These applications can provide support in three types of situations, displaying information and services to a user, automatically executing a service for a user and finally associating the context with the information to facilitate later retrieval [20].

Currently, most context-aware applications in the manufacturing industry focus on creating intelligent information for factory workers, allowing them to respond to manufacturing processes based on this information [22]. These applications have the ability to examine their environment and react to changes in it [23].

In this way, context-aware applications can play an important role in industry by assisting employees in decision-making and improving the performance of factory operations. This assistance is provided by the application, which understands in real time the context of the factory (including tools, machines, parts, products, information relating to the planning of manufacturing processes, among others), processing this information and, based on it, making relevant information/services available to employees [15].

Context-aware applications or systems can and are being integrated with various technologies in order to improve and increase their capabilities. These include technologies such as the Internet of Things (IoT) and Cloud Computing, which are being integrated and

used to implement these applications. In I4.0, the integration of these technologies and their implementation in industry aims to improve quality in manufacturing environments.

2.3. Internet of Things

The Internet of Things (IoT) is a subject of great relevance in the technology, politics and engineering sectors [24]. Many organizations are beginning to recognize and value the possibilities and benefits of IoT to improve their operational activities [25].

The term "Internet of things" was first used in 1999 to describe a system in which real-world objects could be connected to the Internet via sensors, however, today, this expression is common to describe situations in which the Internet connection expands to cover a variety of everyday objects, devices and sensors [24].

For some authors, IoT is considered the main technology of the fourth IR. These objects, or "things", equipped with electronics, software and connected to a network, allow for a more direct integration between the physical world and computer systems. In this way, they make it possible to collect data from their environment and exchange it with information systems [3].

The IoT is a growing innovative technology, applicable to various functions and services in various fields [26]. In industry, this technology makes it possible to obtain and analyze information in real time, contributing to the development of new intelligent applications [25]. Integrated with appropriate systems in industry, the IoT provides efficient services for controlling and managing business processes and resources, thus improving the production processes of manufacturing systems [26].

An era is predicted in which billions of sensors will be connected to the internet. However, processing all the data collected by these sensors is not practical. For this reason, understanding the context will play a crucial role in deciding which data needs to be processed. Context-aware computing also plays a significant role in analyzing, interpreting and understanding this data [27].

2.4. Microservices

Microservices are a growing trend in software architecture. They were inspired by Service Oriented Computing (SOC) and are small applications that can be implemented, scaled and tested independently [28]. They deal with complexity by breaking down large systems into independent services [29].

The microservices approach advocates the decentralization of processes, where each one is designed to develop a specific part of the system [30]. Each microservice is designed and operated as an independent, small-scale system, providing access to its internal logic and data via a well-defined network interface [31]. In a microservice architecture, microservices interconnect to perform a specific function [30]. Each microservice exhibits a unique behavior, and the system is formed by combining and coordinating these microservices through messages [29].

The main characteristics of a microservice system are [29]:

- Flexibility: The system has the ability to easily adapt to changes in the environment and support all necessary modifications;
- Modularity: The system is made up of isolated components, each of which is responsible for a specific task, rather than relying on a single component that offers all the functionality;
- Evolution: It is possible to maintain the system while it evolves, adding new functionalities independently.

The use of microservices has several advantages, such as technological diversity (the possibility of implementing different technologies in the same system), independent implementation (the ability to develop and implement a service in isolation, without depending directly on other services in the system), scalability (the ability to expand services as necessary, without affecting the system as a whole) and ease of maintenance (the ability to modify or replace services without affecting the entire application) [32].

3. Context-Aware Application

In this section, the proposed context-aware application called "Context Engine" is presented, including its design and development process. Section 3.1 discusses the scenario in which the application operates, as well as its responsibilities. Section 3.2 discusses the structure of the system called "Context Aware Factory of Future System" (CA-FoFS), a context-aware information system in which the Context Engine is integrated. Section 3.3 presents the architecture developed for the Context Engine. Subsequently, Section 3.4 details the choice of technology for implementing the applications, as well as how the application integrates and communicates with other systems. Finally, the process of developing the applications throughout the project is detailed in Section 3.5.

3.1. Scenario

The application to be developed integrates the CA-FoFS system, designed specifically for Continental Advanced Antenna (https://cotecportugal.pt/associates/continental-advanced-antenna-portugal/ (assessed on 21 March 2024)), a company that manufactures radio frequency devices for the automotive industry. As such, the data with which the application aims to operate comes from this company.

A variety of data is available to the application, including information on production line stoppages, antenna production per hour, production plans, workers' schedules and their assignments on specific lines, as well as the components needed to produce a product, among others. All this information and context crucial to the operation of the application is stored in databases. It is important to note that the application to be developed does not have direct access to this information. To obtain the necessary data, the application has to make requests to an integration layer via an API. In addition to the information accessible by the integration layer, the Context Engine can also receive data from other applications or devices that are part of the CA-FoFS system it integrates with.

At the Continental factory, operators are responsible for different production lines. During the course of their work, various problems can arise that affect normal manufacturing processes, such as machine breakdowns, material shortages and unexpected stoppages. Continental therefore seeks to solve these problems in order to improve production efficiency.

The main problems that this work aims to solve are requests for missing labels and material, which are currently made by operators in person or by telephone, often having to leave their workstations; the provision of warnings regarding problems on production lines, such as unplanned stoppages or production deviations; and the provision of contextual information to assist workers in making data-based decisions, through the provision of reports or contextual information.

The application to solve these problems works in integration with other services or applications in the Ca-FoFs system to provide contextual services to help workers. Among them, there is a smartwatch that is responsible for carrying out the task of requesting shortages of labels and materials on the production lines, and a virtual assistant in charge of providing warnings regarding unplanned stoppages and production deviations on the production lines, as well as other contextual services, such as providing reports, whose information will be provided by the context-aware application developed.

Therefore, after several meetings with the project members and considering the main problems that the company's coordinators, supervisors and operators face on a daily basis during the production process, the following functionalities were defined that the context-aware application should perform in integration with the CA-FoFS system:

- Identify data changes in data classes in a database from API requests made to the integration layer;
- Identify situations considered urgent or important when analyzing data;
- Send alerts/notifications to other applications/services;

- Maintaining its own data repository to store information received by other applications, recording data that may be needed to verify the proper functioning of the application or for other types of information;
- Process data to acquire contextual information for other services in the global system in which it is inserted and provide an access point to this information.

3.2. CA-FoFS Architecture

The "Context Aware Factory of Future System" (CA-FoFS) is based on a layered architecture, adopting a service/event-oriented approach, and employs context awareness to dynamically support factory users. This layered approach enables the provision of contextual information, integration with various data sources, and supports multiple applications for end users, accessible through various devices [33].

The CA-FoFS architecture is depicted in Figure 1 and consists of five independent layers: the devices layer, applications layer, services layer, integration layer, and lastly, the data layer.

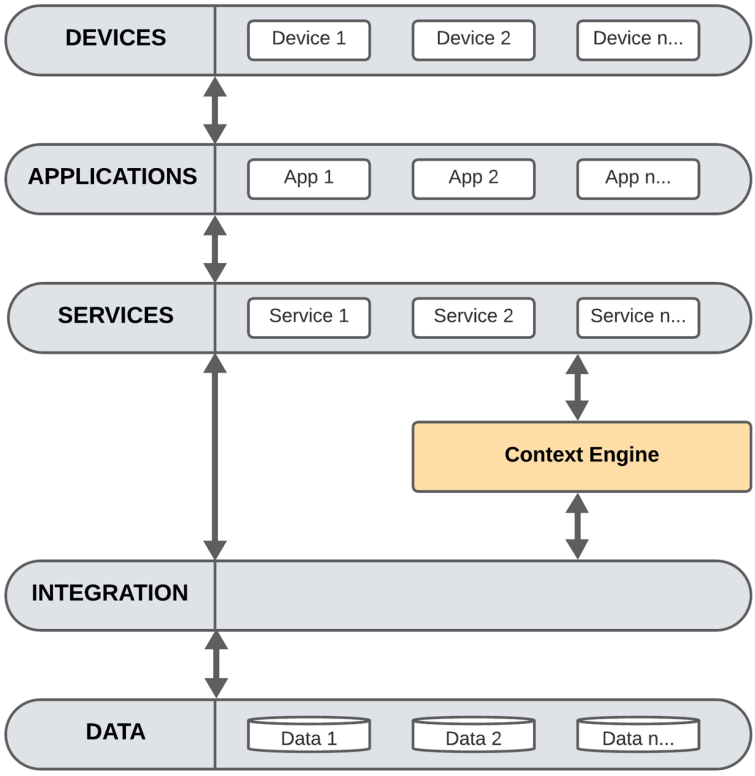

Figure 1. CA-FoFS architecture.

The devices layer facilitates access to contextualized information for users, thus helping them to make decisions. This layer contains various devices that can access the applications in the *application layer*.

The applications layer contains all the applications that can be used by users, and these applications use the services of the *service layer* to be able to perform their information-providing functions.

The services layer is responsible for providing the functions used by applications. These services can access data through the integration layer or the Context Engine.

The Context Engine is the context-aware application proposed in this article. This application acts as an intermediary between the application services and the context data that is accessed from the *integration layer*. Its responsibility is to monitor changes in context and, if it identifies changes that are considered important, to generate alerts/notifications. In ad-

dition, it is responsible for various functionalities for disseminating contextual information based on parameters such as users involved or locations in the factory, among others.

The integration layer has the responsibility of providing an access point to the data available in the data layer, through requests for access to that data, and normalizing it into a common format.

The data layer contains all the data sources from which services and applications can obtain a variety of information. This layer contains company data, stored in databases or other data repositories. This data covers a wide variety of information, including information on production lines, workers and so on, as discussed in Section 3.1.

3.3. Context Engine Architecture

This section proposes a solution for the Context Engine architecture, considering the presented scenario. The proposal aims to provide a flexible, easily scalable, and maintainable solution since the application is in the prototype phase and subject to future adaptations based on new system requirements. Moreover, it should have the capability to integrate with other systems.

Based on previously conducted research on context-aware applications, which is presented in [34], a context-aware application should feature a mechanism capable of acquiring, building, reasoning and disseminating contexts/data. To simplify these tasks, they have been divided into independent components, with each component responsible for executing its respective functions. This results in an approach that facilitates the development and independent maintenance of each component's functionalities.

The proposed architecture is subdivided into three components/applications: Context Acquisition (responsible for detecting and acquiring contexts), Context Builder (responsible for constructing relevant information for the system), and Context Server (responsible for reasoning and disseminating contexts). In addition, the application has its own database called Context DataBase for storing the application's contexts and other relevant information.

Figure 2 illustrates the architecture of the *Context Engine* and its relationship with other parts of the CA-FoFS system.

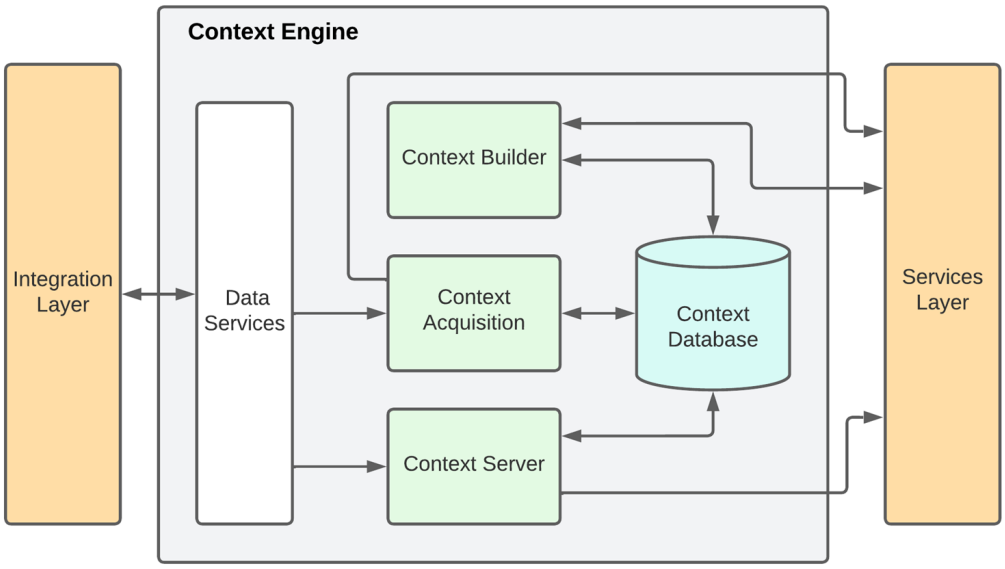

Figure 2. Context Engine Architecture.

The integration layer plays a crucial role as an intermediary in connecting data sources to the *Context Engine*. Its responsibility includes providing authorized data to this application, offering an access point, and normalizing them into a common format.

DataServices is an embedded interface in the architecture designed to assist Context Engine components. Whenever these components need access to company data, they use

this interface, which manages information requests directed to the integration layer. Within this interface, the necessary logic is defined to fulfill these requests.

The Context Acquisition component is responsible for querying the integration layer and checking for changes in the data of certain classes. Periodically, this component will make requests to the integration layer through the DataServices interface. When it identifies modifications in the data, the component must assess the relevance of this data. This process is fundamental for identifying important events immediately and triggering alerts when necessary. These alerts are sent to the "Notification and Alert System" (NAS), a notification and alert service located in the services layer.

The Context Builder has the responsibility of receiving data from other applications in the CA-FoFS system or from IoT devices on the factory's production lines. This data is transmitted intermediately between these applications or devices and the Context Engine through the services layer. Subsequently, this data is processed and stored to obtain relevant information for the system, potentially triggering alerts to the NAS when necessary. Additionally, this component is responsible for removing old data if it is no longer relevant to the system and is only consuming memory resources.

The Context Server assumes logical functions related to application data, responsible for acquiring high-level contextual information through data processing functions. These data are acquired through the integration layer, using the DataServices interface, as well as from the application's own database. Additionally, this component distributes contextual information to stakeholders, providing access to context data. It offers a set of query and information services that can be used by CA-FoFS system applications to obtain relevant data.

The Context Database is the dedicated database for the Context Engine, where essential data for its operation is stored. These data come from the Context Acquisition and Context Builder components. The application uses this data in various functionalities required for system components, as well as for recording information, such as histories of events within the application.

The services layer is part of the CA-FoFS system and assumes the responsibility of providing functions used by applications. These functions have the ability to request information from the Context Server to provide users with a variety of context-based data. Information is delivered through various visualization devices, such as smartwatches, tablets, and others. Devices in the factory environment can also communicate with the Context Engine from the services layer, allowing the sending of new contextual information to the Context Builder. This information is subsequently processed and stored by this component.

3.4. Implementation and Integration Technologies

The selected technology to implement the Context Engine is a microservices' based approach. This decision was supported by the widespread popularity of the technology, as evidenced in the consulted articles and recommendations from their authors [34]. Additionally, microservices exhibit crucial characteristics such as flexibility, modularity, and independent evolution [29], considered highly important for application prototypes due to their adaptability to changes and independent evolution.

Another crucial feature influencing the selection of this technology was the ability to perform implementations through APIs. This facilitates interaction and integration among different software components, enabling efficient communication between various services, a key requirement for this application.

To establish communication and integration with external applications or devices, and considering that the application was developed using microservices, the approach chosen to perform this task was to use the REST API programming interface. This choice enables integration between the various components, using the HTTP protocol to carry out communication tasks. This approach allows different components of the system to commu-

nicate efficiently and flexibly, facilitating the exchange of information and collaboration between services.

Figure 3 demonstrates the integration and communications used by the application.

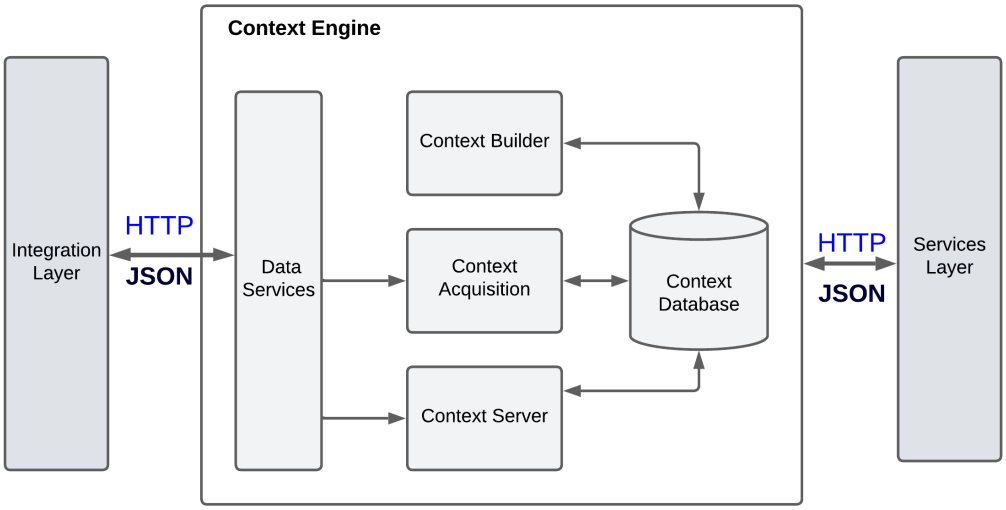

Figure 3. Integration and Communication with External Applications.

After the development of the *Context Engine* architecture, it was observed that its internal components do not need to communicate with each other. They will only exchange data with external applications or with the application's own database. Therefore, it was not necessary to implement communications or integration between these components.

The communications that the application establishes with other external applications are as follows:

1. The application can receive information through the integration layer. To do this, the components that need this data (Context Server and Context Acquisition) must use the DataServices interface. This data is obtained via HTTP GET requests and the information is received in JSON format. In addition, the application can receive data from other applications in the CA-FoFS system, an operation that takes place in the Context Builder. This receives HTTP POST requests with data in JSON format, managing this data according to the service task and returning an HTTP status code indicating the result of the request received.
2. The application can also send information to other external applications. For this to happen, the application must receive a request for information, which is the responsibility of the Context Server component. This receives HTTP GET requests from the service layer, providing an HTTP response with the content of the response in JSON format and the HTTP status code, according to the functionality of the service and the input parameters received. In addition, the application can send alerts to the NAS, which are sent via HTTP POST requests with the information in JSON in the content of the message.

3.5. Development of the Applications

To enable testing and validation of the Context Engine applications during their development, ensuring no interference with the normal operation of the Continental Advanced Antenna company's database, two applications were exclusively developed for testing purposes. These applications simulate the company's database and the integration layer with which the application needs to integrate to obtain this data.

The first application, called ContinentalTestDb, was developed in a web environment using ASP.NET Core MVC and the C# language. Initially, a simulated database was created to replicate the structure of the company's database, as shown in Figure 4. Subsequently,

the application was integrated with this simulated database. Its main objective was to simplify the tasks related to data insertion and manipulation, incorporating logical rules to ensure database consistency. This approach provided a controlled and flexible environment for testing, allowing different scenarios to be explored and the system's behavior to be validated against different sets of data.

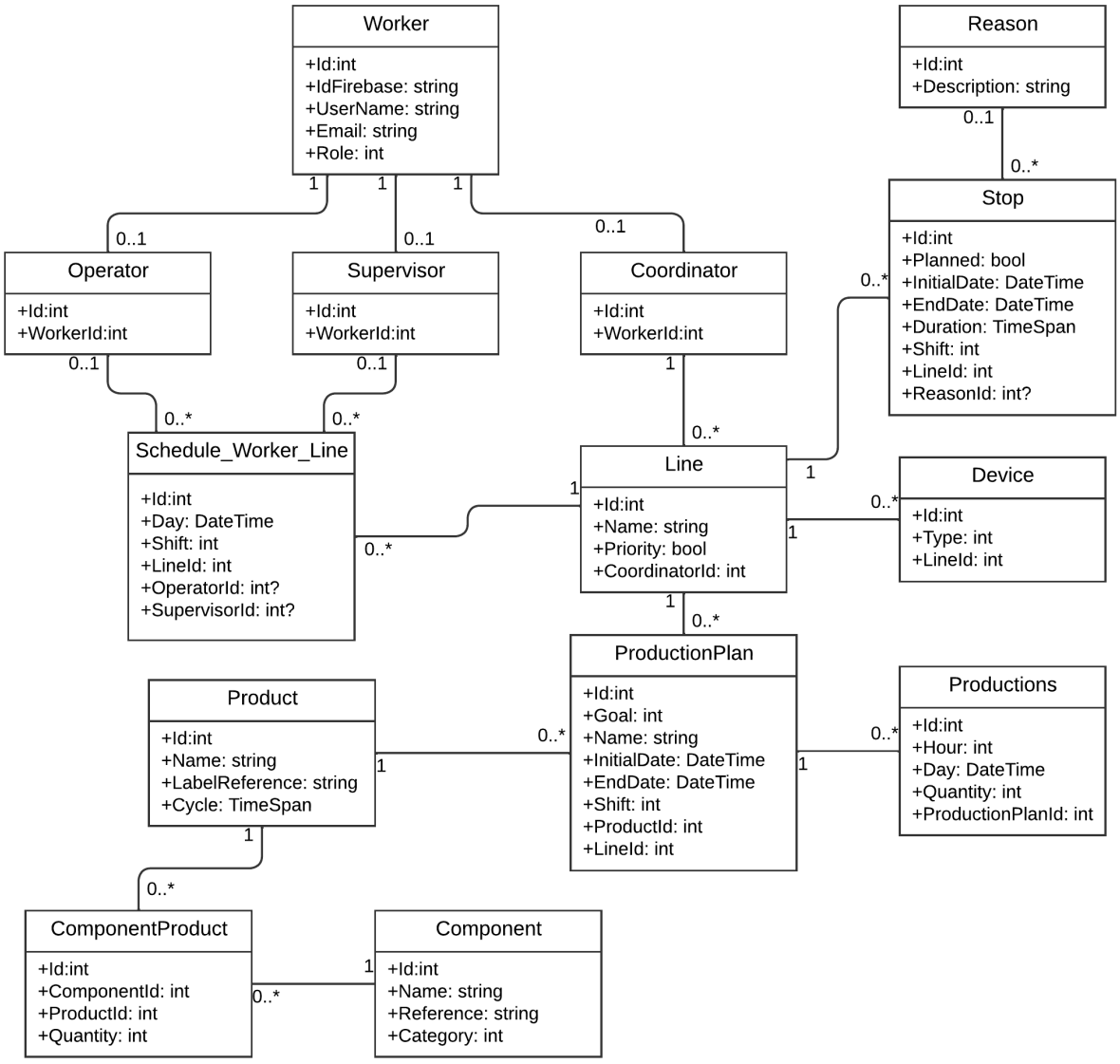

Figure 4. Class Diagram of the Experimental Data Model.

The second application, named *ContinentalTestAPI*, aimed to simulate the *integration layer* of the CA-FoFS architecture. It was developed as a web API using ASP.NET Core in C#. This application replicates the expected behavior of the *integration layer*, allowing for more reliable testing of the *Context Engine* components. This is because these components already interact seamlessly with an application similar to the *integration layer*, making requests for information to an API and receiving information in JSON format.

The development of the *Context Engine* applications began at a later stage. The Context Acquisition was designed as a console application in C#. During its development, a crucial challenge arose in determining how to check for changes in the company's database accessed through requests to the *integration layer* via API. Initial research revealed that custom methods are often required to determine changes in data, involving triggers,

adding columns to store last modification dates, or even storing changes in additional tables [35]. However, none of these options were viable, as it was not intended to interfere with or modify the company's data system, and permission to do so was not given. The solution found was to make data requests to the API and store this data in the Context Engine's own database, allowing the comparison of information in subsequent requests to detect changes. The subsequent development of the functionalities of this application was relatively straightforward. With the identified changes, it was only necessary to check their relevance and, if affirmative, send the corresponding alert through an HTTP POST request to the NAS. Table 1 describes the various services and features that Context Acquisition uses to perform its functions.

Table 1. Context Acquisition services.

#	Service Name	Description
1	Checking for Data Changes	Responsible for verifying changes in data related to stoppages and productions in the company database, through requests to the integration layer via API.
2	Update items	Responsible for adding or updating items verified as changes to data in the company's database in the application's own database.
3	Check importance	Checks whether the data detected is considered relevant to trigger the sending of alerts/warnings. The relevant data for triggering warnings are those relating to unplanned stoppages that have occurred and data relating to new production that has occurred with a production deviation below the expected number.
4	Send alerts	Sending alerts to the NAS regarding urgent stoppages or production deviations.

Both the *Context Builder* and the *Context Server* were developed as web APIs using ASP.NET Core in the C# language. The *Context Builder* receives information from other applications through HTTP POST requests, processes this information, and stores it in the database if necessary. This component includes information reception services, such as receiving data about the absence of a component on a production line. This information is sent by an IoT device (a smartwatch) available to the company's operators on the production lines. Based on this information, the *Context Builder* sends alerts to the NAS. Another responsibility of this component is the removal of old data from the database if it is no longer needed for the system. To perform this task, background tasks were implemented in the application, which are executed at predefined time intervals to check for data that is no longer relevant, subsequently proceeding with its removal. Table 2 describes the services and functionalities provided by Context Builder.

Through an API, the *Context Server* provides information via HTTP GET requests, which are sent by the services in the CA-FoFS system's *service layer* when they need this information to operate. The logic for responding to these information requests is implemented in each API service. The Context Server component provides a diverse set of query and information services intended to be used by other applications to obtain relevant data. To respond, these services use information contained in the company's databases from the *integration layer* or information contained in the application's own database. These services include information about what is happening on the factory's production lines, such as the products currently in production or the lack of a component on that line. In addition, it offers services for requesting information stored in the database itself, such as the history of alerts that Context Engine has sent to the NAS or requests for information made by certain users to the CA-FoFS system. In total, the application has 14 independent information services, detailed in Table 3, which highlights their names, input parameters and corresponding descriptions.

Table 2. Context Builder services.

#	Service Name	Description
1	Receive and store data	This component provides API services for receiving and storing data in the database. One example is the reception of Requests, which consist of data related to requests for information made by users to other services in the CA-FoFS system and which are stored in the database.
2	Material replacement management	This component provides API services for managing material replenishment information on production lines. It currently includes services related to the replenishment of labels and components needed for production. When an operator requests the replenishment of materials, this component receives the request, stores the information in the database and sends an alert to the NAS. Once the replenishment is complete, the information about the missing material is deleted from the database. In this way, supervisors can use the application to view a list of missing materials on the production lines and take the necessary action.
3	Send alerts	Sending alerts to the NAS regarding requests to replace components or labels on production lines.
4	Old data removal	This component has a service that operates in the background and removes old data from the database that is no longer needed by the application.

Table 3. Context Server services.

#	Service Name	Input Parameter(s)	Description
1	Device Info	Device ID	It provides information about the device, including its type (wearable, tablet, etc.). If the device is associated with an operator, it provides information about the production line the device is associated with, the product currently in production, its label, the list of missing components on the line that need replacing and the work shift. If the device is associated with a coordinator, it provides information about the coordinator and the production lines for which he is responsible.
2	Operator Info	Worker's firebase ID	It provides information about the operator and the production lines on which he is working on the current day, along with their working schedules.
3	Stops Info	Start date * End date * Planned *	It provides information on the stops that have taken place according to the dates entered, depending on whether the stops were planned or not.
4	Line Info	Production line ID Start date * End date *	It provides information relating to the production line, providing data on the stoppages and productions that have taken place on the line, according to the dates entered. It also provides information relating to the coordinator responsible for the line and the products that are or were being produced during those dates.
5	Supervisor Info	Worker's firebase ID Day *	Provides information about the supervisor and the production lines he was supervising on the day entered, along with the respective working schedules, according to the request.
6	Get Productions Info	Production line ID Start date * End date *	Provides a list of productions that have taken place according to the dates entered.
7	Get Components Device Info	Device ID	Provides the list of components being used on the production line to which the device is associated.
8	Product Info	Production line ID	Provides information on the product being produced on the production line when the request for information was made.

Table 3. *Cont.*

#	Service Name	Input Parameter(s)	Description
9	Coordinator Info	Worker's firebase ID	Provides information on the coordinator and the production lines for which he is responsible, according to the request.
10	Notification Recommendation	Notification type Worker ID	Provides, if possible (if the employee already has working schedule defined for that day), a date to send them a notification regarding certain information, such as sending a graph or other information. If this is not possible, it sends the day and part of the shift on which the notification should be sent.
11	Get Missing Components	Production line ID * Component ID * Day *	It returns information on the missing components, the respective production lines affected and the date on which the request to replace the component was made, depending on the search parameters.
12	Get Missing Labels	Production line ID * Label reference * Day *	Returns information on the missing labels, the respective production lines affected and the date on which the label replacement request was made, according to the search parameters.
13	Get Alerts History	Alert type * Start date * End Date *	Returns the history of alerts sent by the application, according to the search parameters.
14	Get Requests History	Order type * Day * worker ID *	Returns the history of requests for viewing the information requested by users, according to the search parameters.

Parameters marked with * are optional.

4. Experimental Findings

This section presents the experimental results obtained from tests carried out on the *Context Engine*, the developed context-aware application, with the goal of demonstrating the usefulness and usability of this application. Section 4.1 presents the tests carried out on the application and presents some observations on this process. In Section 4.2, some case studies are presented, demonstrating how the *Context Engine* services are integrated with other services in the CA-FoFS system, such as the virtual assistant and the wearable (smartwatch), providing a detailed and contextualized view on the practical application of the proposed solution.

4.1. Tests

The tests conducted on the Context Engine, namely, its components—*Context Acquisition*, *Context Builder*, and *Context Server*, occurred in a laboratory environment. This assessment included unit tests, where the application services were evaluated in isolation, and functional tests, where the operation of the Context Engine was observed when integrated with services simulating the integration and services layers. In other words, it involved fictitious data and applications created to test its components.

The unit tests were conducted using the "xUnit.net" tool, which is a framework for creating unit tests for the .NET Framework. For each application service, these tests covered functionalities related to the logic of the developed services, and their development followed the following protocol:

- Preparation: Configure the necessary dependencies (controllers, interfaces, classes, etc.) using fake objects, simulate the required information (stops, productions, etc.), and configure the behavior of some functions to return the desired result for the test (for example, OK or BadRequest).
- Action: Create instances of the class of methods to be tested, passing the necessary dependencies, instantiating the static information to be used, invoking the respective methods, and capturing the outputs.
- Verification: Verify the correct execution, ensuring that the methods that modify the data in the database were called the expected number of times and confirming whether the returned information or console output is as expected.

The mentioned fake objects, created using the *FakeItEasy* library, facilitate testing at this stage. All objects will be fake, and subsequently, the library itself determines whether they are mocks or another type.

In total, 61 unit tests were conducted on the components of the *Context Engine*: 10 on *Context Acquisition*, 11 on *Context Builder*, and 40 on *Context Server*. All tests were approved, meaning they were successful.

Regarding functional tests, the functional requirements to be tested were defined, and later, the method for testing this functionality of the application was determined. After conducting the necessary tests, it was recorded whether this functionality was executed as expected. The functional tests conducted in the application are detailed in Table 4, indicating to which applications they were applied, their description, and the test results. All functionalities for which the components of the Context Engine are responsible passed the tests, confirming that the applications are functioning as expected.

Table 4. Functional tests.

#	Description	Application(s)	Results
1	Identification of changes when inserting new data into the simulated database and recognition of urgent or important situations during the analysis of this data.	*Context Acquisition*	Working correctly
2	Verification of whether, after identifying changes in the data, there is an addition or update in the Context Engine database.	*Context Acquisition*	Working correctly
3	Verification of the sending of alerts/notifications to the NAS when necessary and confirmation in the history of the Context Engine database regarding the recording of the sent alerts.	*Context Acquisition* *Context Builder*	Working correctly
4	Verification of the application's ability to manage/clean data by eliminating those older than deemed valid after their addition to the database.	*Context Builder*	Working correctly
5	Verification of the correct operation of the services by executing API methods with various data, analyzing different situations.	*Context Builder* *Context Server*	Working correctly

Figure 5 presents an example of the Context Acquisition in operation when tests 1, 2, and 3 from Table 4 were applied to verify its functioning. In this test, two productions and two stops were added to the database through the ContinentalTestDb application; for each dataset, one object requires sending an alert, while the other does not. As can be seen in Figure 5, in the red square, the application identified all four data changes.

In the blue square, it is presented what the application performed after these checks, showing that tests 1, 2, and 3 were approved. In other words, all four data points were added to the *Context Engine* database, the application recognized urgent or important situations during the analysis of this data, and the corresponding alerts were successfully sent for both cases that should occur.

These tests were not only verified by the console outputs presented in Figure 5, but it was also confirmed in the Context Engine database whether the identified data changes were added, and if information about the respective alert submissions was also recorded. After this verification, it was confirmed that the application was functioning correctly.

In order to conduct test 3, presented in Table 4, in the *Context Builder*, the approach was similar. Information about a missing component in a production line was sent through an HTTP POST request to an API microservice. After this submission, it was verified whether the application processed the information correctly, sent the corresponding alert, and recorded the information in the database. This test was also approved, demonstrating that the functionality is operating correctly.

Test 4, presented in Table 4, conducted in the *Context Builder*, was executed as follows: data was added to the database with dates older than those considered valid by the application. After this insertion, the necessary time was waited for the application to clean up old data. Subsequently, it was verified in the database whether this data was successfully removed. After this verification, it was confirmed that this functionality was operating correctly.

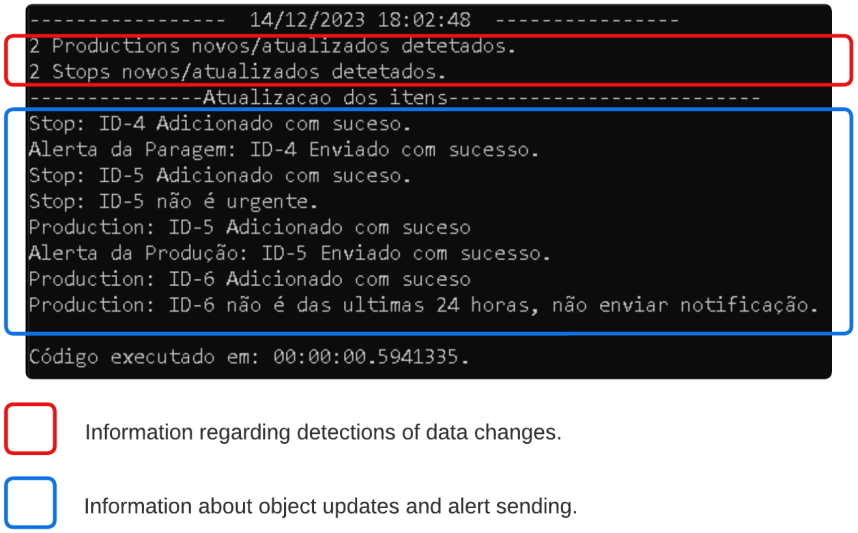

Figure 5. Context Acquisition Execution Example for Tests.

Finally, test 5 presented in Table 4 was conducted in the *Context Builder* and *Context Server*. In this test, each service of the APIs of these applications was evaluated by inserting various input parameters into these services and checking if the response was as expected. For the information services of the *Context Server*, a variety of data was added to the database simulating the company's database to test various scenarios and the expected responses by the application. All tests conducted were approved, confirming that, in these cases, the application is operating correctly.

Figures 6 and 7 illustrate a test of this kind. Two requests were made to the *Coordinator-Info* service of the *Context Server*. In the first one, shown in Figure 6, a request was made to the service, sending an idfirebase of a worker who is not a coordinator type. The application responded with the worker's data, accompanied by a "NotFound" response, indicating that it could not identify the coordinator. In the second test, represented in Figure 7, a request was made by sending an idfirebase of a worker who is a coordinator. The service returned "OK," indicating that the information was successfully obtained, along with the respective worker's information and the lines they coordinate. These two test examples demonstrate that this application service is functioning as intended. The other functional tests for the services of this application were executed similarly.

These tests demonstrate the capabilities of the *Context Engine* and show that it is working correctly. It is, however, important to assess how this solution could be applied in a real environment. The following section aims to demonstrate the potential and advantages of such practical application.

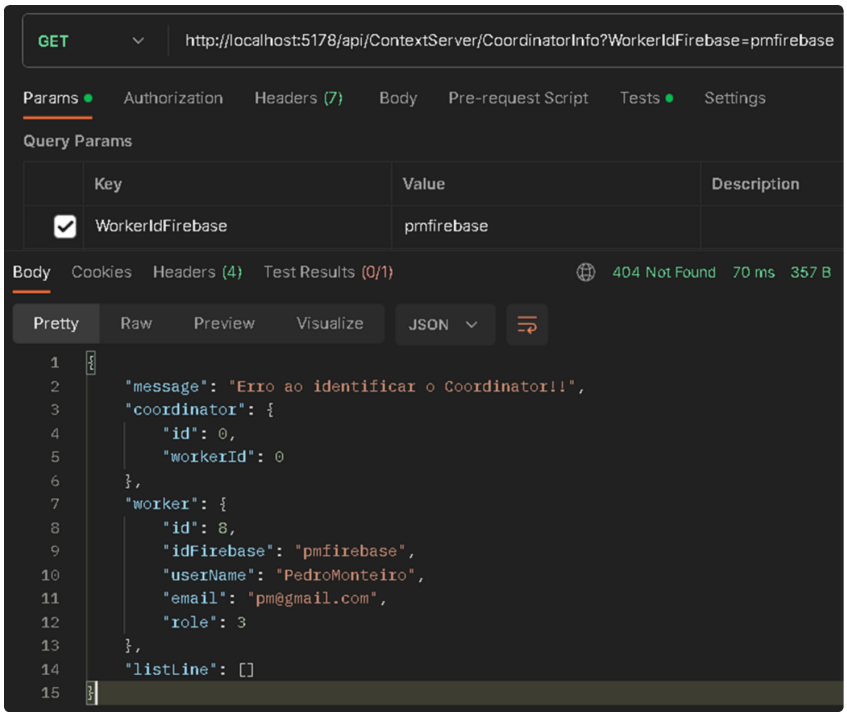

Figure 6. Search test for a worker who is not of the coordinator type.

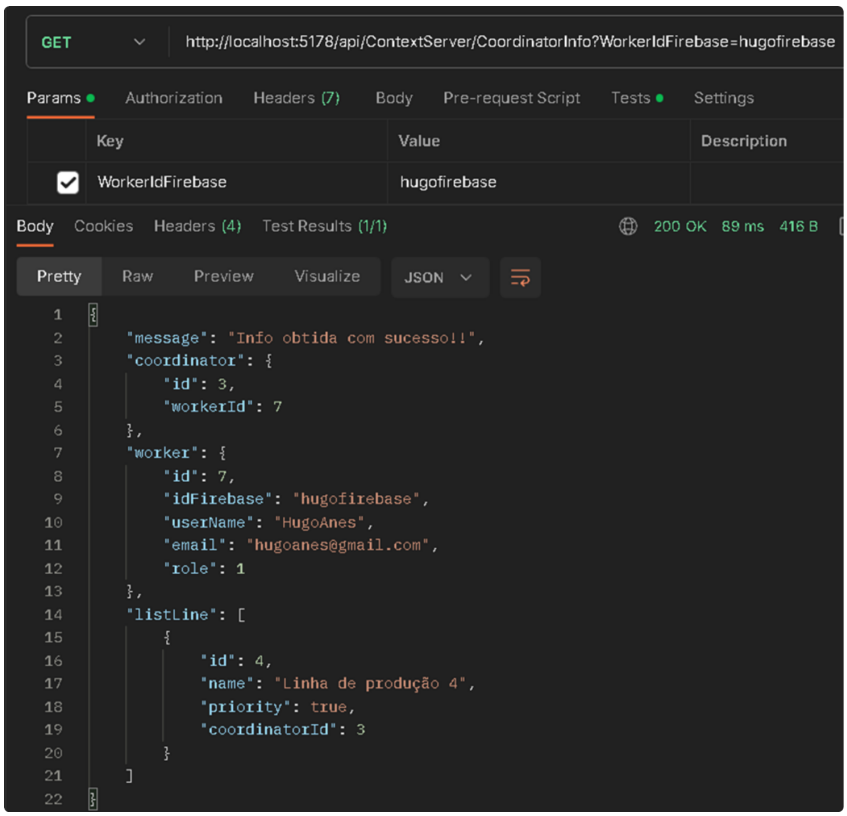

Figure 7. Search test for a worker who is a coordinator.

4.2. Case Studies

The purpose of this case study section is to provide a more detailed and contextualized view of *Context Engine's* specific contribution to the actual factory. This section presents examples of *Context Engine* services that are used by other services in the CA-FoFS system, such as the virtual assistant and the wearable (smartwatch). Throughout this section, the

figures presented for the virtual assistant and smartwatch are pictures of these applications working with static data or mockups. The case studies presented in this section are intended to demonstrate examples of data requests that other applications can make to *Context Engine*, examples of responses that the application can provide to these requests and examples of how these services use this information. The case studies are presented below as follows.

4.2.1. Case Study 1: Sending Urgent Stop Alerts and Making Them Available through the Virtual Assistant

This case study shows an example of how the virtual assistant uses the information from the stop alerts sent by the *Context Engine* to the NAS. *Context Acquisition* is responsible for detecting new stoppages. When new data relating to this information is entered into the company's database, it is detected by the application, which studies the data. If the data is considered important, alerts are sent to the NAS.

Figure 8a shows an example of the detection of two breakdowns and the sending of alerts to the NAS and Figure 8b shows the application saving the data relating to the sending of alerts in its database.

```
----------------  09/02/2024 01:22:17  ----------------
0 Productions novos/atualizados detetados.
2 Stops novos/atualizados detetados.
--------------Atualizacao dos itens--------------------------
Stop: ID-6 Atualizado com suceso.
Alerta da Paragem: ID-6 Enviado com sucesso.
Stop: ID-7 Atualizado com suceso.
Alerta da Paragem: ID-7 Enviado com sucesso.

Código executado em: 00:00:00.0402638.
```

a) Stop detection


```
{
    "id": 34,
    "typeOfAlet": 1,
    "alertSuccessfullySent": true,
    "errorMessage": "",
    "alertMessage": "Paragem urgente detetada, Id-6, LineId-1, Razão - Problem 36 :
        Quality on machine 2.",
    "alertDate": "2024-02-09T01:22:17.6532447"
},
{
    "id": 35,
    "typeOfAlet": 1,
    "alertSuccessfullySent": true,
    "errorMessage": "",
    "alertMessage": "Paragem urgente detetada, Id-7, LineId-1, Razão - Problem 12 :
        Soldering issue on machine 1.",
    "alertDate": "2024-02-09T01:22:17.6738052"
}
```


b) Registration of information for sending alerts

Figure 8. Detecting Stoppages and Recording the Sending of Alerts in *Context Acquisition*.

With the respective information that has been sent to the NAS, the virtual assistant will make these warnings/alerts available. Figure 9 shows an example of this assistant providing this information.

The alerts sent by *Context Acquisition* about production deviations operate in the same way. The information on stoppage alerts and production deviations provided by the virtual assistant allows workers to monitor what is happening on the production lines more actively. From this information considered critical by the application, users are informed about what has happened on the production lines and can take corresponding measures if necessary.

Figure 9. Alerts Received Page in the Virtual Assistant [36].

4.2.2. Case Study 2: Material Replacement Orders from Production Lines

The operators of the factory's production lines are equipped with smartwatches. These wearables are integrated with the CA-FoFS system and have two important functions for the production line, which need data from the *Context Engine* in order to work. These functionalities are the request for labels or the request for replacement components, which are shown in Figure 10.

Figure 10. Material Replacement Request functionality on the Smartwatch.

With regard to the request for labels, operators make this request on the smartwatch and it is forwarded to the system. To find out which labels need to be replaced, the system makes an information request to the Context Engine's *DeviceInfo* service (service 1 shown in Table 3). This service, based on the device ID, provides the label reference of the product currently in production on the production line with which the device is associated. With this information, the system sends a label replacement notice with this reference to the entities responsible for the replacement. Figure 11 shows an example of an information request to the Context Engine's *DeviceInfo* service, through which this reference can be obtained. This is obtained via the tagReference parameter, as can be seen in the figure.

Request URL

```
http://localhost:5178/api/ContextServer/DeviceInfo?deviceId=1
```

Server response

Code	Details
200	Response body


```
{
  "message": "Info obtida com sucesso!!",
  "type": "Weareble-Operator",
  "line": {
    "id": 1,
    "name": "Linha1",
    "priority": true,
    "coordinatorId": 1
  },
  "productName": "Antena Parabólica 1",
  "tagReference": "AP789",
  "listMissingComponentes": [],
  "workShift": 1,
```


Figure 11. Example of *DeviceInfo* Service Response to an Information Request.

The other functionality is the request for replacement components for the production line. In this functionality, the operator clicks on the materials request shown in Figure 10 from the smartwatch. After this click, the device displays a page where you can select the components that need to be replaced on the production line. In order to provide this list of components, the device uses the *GetComponentsDeviceInfo* service (service 7 shown in Table 3). Based on the device ID, the *Context Engine* provides a list of components, which are the components used in the production of the product currently being manufactured on the production line with which the device is associated. Figure 12a shows an example of the application providing this list of components and Figure 12b shows a mockup of this list displayed on the wearable.

```
http://localhost:5178/api/ContextServer/GetComponentsDeviceInfo?deviceId=1
```

Server response

Code	Details
200	Response body


```
{
  "message": "Info obtida com sucesso!!",
  "listComponents": [
    {
      "id": 11,
      "name": "Parafuso A2",
      "reference": "OTB-340 ",
      "category": 1
    },
    {
      "id": 10,
      "name": "Parafuso A1",
      "reference": "CTS-520 ",
      "category": 1
    },
    {
      "id": 9,
      "name": "Filtro BS1",
      "reference": "CCX-550 ",
      "category": 1
    }
  ]
}
```


a)Context Engine provides the list of components

b)Example of a mockup of the smartwatch's functionality for requesting replacement components

Figure 12. Component replacement order functionality.

The operator can then select the components that need to be replaced and the corresponding replacement alerts are sent to the production line where the device is associated.

These features offered by the *Context Engine* to the CA-FoFS system help operators by allowing them to request replacement components or labels simply and quickly, without the need to leave the production line.

4.2.3. Case Study 3: Provision of Contextual Information for Creating Reports by the Virtual Assistant

Another task that the services provided by *Context Engine* have is to assist the virtual assistant in creating and making available PDF reports for sending by email. *Context Engine* helps this virtual assistant by providing its contextual services, which are used to generate reports based on this information. These reports are then made available to factory workers by this assistant.

Figure 13 shows a virtual assistant service that uses *Context Engine* services to create reports. This service creates a report on what happened on the production lines on a certain day. In this case, it provides information about the productions that took place, the stoppages that were recorded and presents a graph about the time spent on productions with these stoppages.

Figure 13. Virtual Assistant Report creation functionality [36].

For this virtual assistant service to be able to present this information, it needs to make requests for information to the *Context Engine*, more specifically to the *LineInfo* service (service 4 presented in Table 3). This service, based on the production line ID and search dates entered, provides the productions and stoppages that occurred on that production line. With this information, the virtual assistant can create the report.

The *Context Engine* provides other information services that can be used by the virtual assistant and the applications of the CA-FoFS system. However, these three presented case studies manage to provide a more detailed insight into the contribution of the Context Engine to the company, through some practical illustrative cases. Overall, the *Context Engine* is a fundamental application for building the CA-FoFS system. This application provides various context services that offer essential information to the applications of this

system so that they can operate, besides providing information to users to assist them in decision-making.

5. Conclusions and Future Work

This article presents and discusses the development of a context-aware application called *Context Engine*. Initially, a theoretical study on this type of application is presented. This study enables reaching the conclusion that contextual applications can perform various functions and that, when applied in the manufacturing industry, they aim to generate intelligent information for workers. Such information enables them to respond to manufacturing processes in an instructed and informed way.

The Context Engine architecture was designed taking into account the operational scenario, the functions it performs, the previous study on this type of application in the article [34], as well as the CA-FoFS architecture. Divided into three main components—*Context Acquisition*, *Context Builder* and *Context Server*—according to the different capabilities they have, the architecture makes it possible to modify and add functionality independently, which is crucial given that it is still in a prototype state.

By using microservices technology, a choice justified by its flexibility, modularity and evolution, the application is able to adapt to changes and incorporate new features independently. In addition, the use of this technology allows efficient integration and communication between services via APIs, which was crucial in the development of the application. Once the applications had been developed, their functionalities were tested in each application in a laboratory environment and showed their correct functioning and development.

Once the applications had been developed, their functionality was tested in a laboratory environment using unit and functional tests. 61 unit tests were carried out on the Context Engine components and all passed, confirming that the services tested behaved as expected and provided responses as predicted. As for the functional tests, all the functionalities provided by Context Engine were tested in each application, integrating them with services that simulate the integration and services layer. All the functionalities were found to be working correctly.

As prospects for future work, there are several tasks that can be incorporated into this application, given the extensive range of functionalities that this type of application is capable of performing. One particularly intriguing and essential initiative for production environments would be the integration of *Context Acquisition* with IoT devices to process data in real time and generate valuable information for the company. Various IoT devices, such as temperature, humidity, pressure and vibration sensors, among others, could be applied in the company. The information from these sensors could be received and processed by *Context Acquisition* and used to monitor environmental conditions in production areas, ensuring that they are suitable for production, for workers or even to carry out preventive maintenance on machines.

In addition, the application's information provision functionalities can be extended. For example, functionalities relating to crucial information such as stock levels, the location of parts in the factory and production estimates could be incorporated. This expansion of functionalities would not only offer more comprehensive support for decision-making, but would also contribute to optimizing processes and improving the overall performance of the production environment.

Finally, security rules need to be defined and implemented in the application to guarantee data integrity and protection. Given the current emphasis on the architecture and functionality of the application, the issue of security has been left open to be dealt with at a later date. It will be necessary to carry out a comprehensive analysis of security measures for future iterations and enhancements of communications between the Ca-FoFs system and the Context Engine, ensuring that they comply with appropriate security standards.

Author Contributions: Conceptualization, A.R. and T.P.; methodology, R.P.; software, P.M.; validation, R.P., R.N. and T.P.; investigation, P.M., R.P. and T.P.; resources, A.R. and T.P.; data curation, P.M. and R.P.; writing—original draft preparation, P.M.; writing—review and editing, T.P.; visualization, R.N.; supervision, A.R. and T.P.; project administration, A.R.; funding acquisition, A.R. All authors have read and agreed to the published version of the manuscript.

Funding: The study was developed under the project A-MoVeR—"Mobilizing Agenda for the Development of Products & Systems towards an Intelligent and Green Mobility", operation n.º 02/C05-i01.01/2022.PC646908627-00000069, approved under the terms of the call n.º 02/C05-i01/2022—Mobilizing Agendas for Business Innovation, financed by European funds provided to Portugal by the Recovery and Resilience Plan (RRP), in the scope of the European Recovery and Resilience Facility (RRF), framed in the Next Generation UE, for the period from 2021–2026.

Data Availability Statement: The datasets presented in this article are not readily available because data is the property of Continental A.A.

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

References

1. Sivakumar, K.; Dhyankumar, C.T.; Cherian, T.M.; Manikandan, N.; Thejasree, P. Requirements for the Adoption of Industry 4.0 in the Sustainable Manufacturing Supply Chain. In *Industry 4.0 Technologies: Sustainable Manufacturing Supply Chains: Volume II-Methods for Transition and Trends*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 185–201.
2. Ferreira, J.J.; Lopes, J.M.; Gomes, S.; Rammal, H.G. Industry 4.0 implementation: Environmental and social sustainability in manufacturing multinational enterprises. *J. Clean. Prod.* **2023**, *404*, 136841. [CrossRef]
3. Bisio, I.; Garibotto, C.; Grattarola, A.; Lavagetto, F.; Sciarrone, A. Exploiting context-aware capabilities over the internet of things for industry 4.0 applications. *IEEE Netw.* **2018**, *32*, 101–107. [CrossRef]
4. Xu, M.; David, J.M.; Kim, S.H. The fourth industrial revolution: Opportunities and challenges. *Int. J. Financ. Res.* **2018**, *9*, 90–95. [CrossRef]
5. Hermann, M.; Pentek, T.; Otto, B. Design Principles for Industrie 4.0 Scenarios. In Proceedings of the 2016 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, USA, 5–8 January 2016.
6. Petrasch, R.; Hentschke, R. Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method. In Proceedings of the 2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE), Khon Kaen, Thailand, 13–15 July 2016; pp. 1–5.
7. Yavuz, O.; Uner, M.M.; Okumus, F.; Karatepe, O.M. Industry 4.0 technologies, sustainable operations practices and their impacts on sustainable performance. *J. Clean. Prod.* **2023**, *387*, 135951. [CrossRef]
8. Olsen, T.L.; Tomlin, B. Industry 4.0: Opportunities and Challenges for Operations Management. *Manuf. Serv. Oper. Manag.* **2020**, *22*, 113–122. [CrossRef]
9. Wieland, M.; Leymann, F.; Schäfer, M.; Lucke, D.; Constantinescu, C.; Westkämper, E. Using context-aware workflows for failure management in a smart factory. In Proceedings of the Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies UBICOMM, Florence, Italy, 25–30 October 2010; pp. 379–384.
10. Flatt, H.; Koch, N.; Röcker, C.; Günter, A.; Jasperneite, J. A context-aware assistance system for maintenance applications in smart factories based on augmented reality and indoor localization. In Proceedings of the 2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA), Luxembourg, 8–11 September 2015; pp. 1–4.
11. Zhong, R.Y.; Xu, X.; Klotz, E.; Newman, S.T. Intelligent manufacturing in the context of industry 4.0: A review. *Engineering* **2017**, *3*, 616–630. [CrossRef]
12. Shi, Z.; Xie, Y.; Xue, W.; Chen, Y.; Fu, L.; Xu, X. Smart factory in Industry 4.0. *Syst. Res. Behav. Sci.* **2020**, *37*, 607–617. [CrossRef]
13. Gubán, M.; Kovács, G. INDUSTRY 4.0 CONCEPTION. *Acta Tech.-Corviniensis-Bull. Eng.* **2017**, *10*, 22.
14. Reza, M.N.H.; Malarvizhi, C.A.N.; Jayashree, S.; Mohiuddin, M. Industry 4.0—Technological revolution and sustainable firm performance. In Proceedings of the 2021 Emerging Trends in Industry 4.0 (ETI 4.0), Raigarh, India, 19–21 May 2021; pp. 1–6.
15. Alexopoulos, K.; Makris, S.; Xanthakis, V.; Sipsas, K.; Chryssolouris, G. A concept for context-aware computing in manufacturing: The white goods case. *Int. J. Comput. Integr. Manuf.* **2016**, *29*, 839–849. [CrossRef]
16. Bang, A.O.; Rao, U.P. Context-aware computing for IoT: History, applications and research challenges. In Proceedings of the Second International Conference on Smart Energy and Communication: ICSEC 2020, Jaipur, India, 20–21 March 2020; Springer: Singapore, 2021; pp. 719–726.
17. Wollschlaeger, M.; Sauter, T.; Jasperneite, J. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Ind. Electron. Mag.* **2017**, *11*, 17–27. [CrossRef]
18. Lasi, H.; Fettke, P.; Kemper, H.G.; Feld, T.; Hoffmann, M. Industry 4.0. *Bus. Inf. Syst. Eng.* **2014**, *6*, 239–242. [CrossRef]
19. Mohamed, M. Challenges and benefits of industry 4.0: An overview. *Int. J. Supply Oper. Manag.* **2018**, *5*, 256–265.
20. Dey, A.K. Understanding and using context. *Pers. Ubiquitous Comput.* **2001**, *5*, 4–7. [CrossRef]

21. Abowd, G.D.; Dey, A.K.; Brown, P.J.; Davies, N.; Smith, M.; Steggles, P. Towards a better understanding of context and context-awareness. In Proceedings of the Handheld and Ubiquitous Computing: First International Symposium, HUC'99, Karlsruhe, Germany, 27–29 September 1999; Proceedings 1; Springer: Berlin/Heidelberg, Germany, 1999; pp. 304–307.
22. Ye, Y.; Hu, T.; Nassehi, A.; Ji, S.; Ni, H. Context-aware manufacturing system design using machine learning. *J. Manuf. Syst.* **2022**, *65*, 59–69. [CrossRef]
23. Schilit, B.; Adams, N.; Want, R. Context-Aware Computing Applications. In Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications, Washington, DC, USA, 8–9 December 1994; pp. 85–90. [CrossRef]
24. Rose, K.; Eldridge, S.; Chapin, L. The internet of things: An overview. *Internet Soc. (ISOC)* **2015**, *80*, 1–50.
25. Breivold, H.P.; Sandström, K. Internet of Things for Industrial Automation—Challenges and Technical Solutions. In Proceedings of the 2015 IEEE International Conference on Data Science and Data Intensive Systems, Sydney, Australia, 11–13 December 2015; pp. 532–539.
26. Lampropoulos, G.; Siakas, K.; Anastasiadis, T. Internet of things in the context of industry 4.0: An overview. *Int. J. Entrep. Knowl.* **2019**, *7*, 4–19. [CrossRef]
27. Perera, C.; Zaslavsky, A.; Christen, P.; Georgakopoulos, D. Context Aware Computing for The Internet of Things: A Survey. *IEEE Commun. Surv. Tutor.* **2014**, *16*, 414–454. [CrossRef]
28. Larrucea, X.; Santamaria, I.; Colomo-Palacios, R.; Ebert, C. Microservices. *IEEE Softw.* **2018**, *35*, 96–100. [CrossRef]
29. Dragoni, N.; Giallorenzo, S.; Lafuente, A.L.; Mazzara, M.; Montesi, F.; Mustafin, R.; Safina, L. Microservices: Yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*; Springer: Cham, Switzerland, 2017; pp. 195–216.
30. Shadija, D.; Rezai, M.; Hill, R. Towards an understanding of microservices. In Proceedings of the 2017 23rd International Conference on Automation and Computing (ICAC), Huddersfield, UK, 7–8 September 2017; pp. 1–6. [CrossRef]
31. Jamshidi, P.; Pahl, C.; Mendonça, N.C.; Lewis, J.; Tilkov, S. Microservices: The journey so far and challenges ahead. *IEEE Softw.* **2018**, *35*, 24–35. [CrossRef]
32. Viggiato, M.; Terra, R.; Rocha, H.; Valente, M.T.; Figueiredo, E. Microservices in practice: A survey study. *arXiv* **2018**, arXiv:1808.04836.
33. Continental FoF (2023) Continental FoF—Continental AA's Factory of Future (utad.pt). Available online: https://fof.utad.pt/ (accessed on 23 April 2024).
34. Monteiro, P.; Lima, C.; Pinto, T.; Nogueira, P.; Reis, A.; Filipe, V. Context-Aware Applications in Industry 4.0: A Systematic Literature Review. In *International Symposium on Distributed Computing and Artificial Intelligence*; Springer: Cham, Switzerland, 2023; pp. 301–311.
35. Controle de Alterações de Dados (SQL Server). Available online: https://learn.microsoft.com/pt-br/sql/relational-databases/track-changes/track-data-changes-sql-server?view=sql-server-ver16 (accessed on 13 November 2023).
36. Pereira, R. Uso de Assistentes Virtuais no Apoio à Gestão de Produção. Master's Thesis, Universidade de Trás-os-Montes e Alto Douro, Vila Real, Portugal, 2023.