A self-adaptive IoT architecture to support intelligent environments

Mateus G. do Nascimento
mateus.goncalo@ice.ufjf.br
Federal University of Juiz de Fora
Juiz de Fora, Minas Gerais, Brazil

José Maria N. David
jose.david@ufjf.br
Federal University of Juiz de Fora
Juiz de Fora, Minas Gerais, Brazil

Mario A. R. Dantas
mario.dantas@ice.ufjf.br
Federal University of Juiz de Fora
Juiz de Fora, Minas Gerais, Brazil

Regina Braga
regina.braga@ufjf.br
Federal University of Juiz de Fora
Juiz de Fora, Minas Gerais, Brazil

Victor Ströele
victor.stroele@ufjf.br
Federal University of Juiz de Fora
Juiz de Fora, Minas Gerais, Brazil

ABSTRACT

Context: Intelligent environments are complex interaction spaces between people, sensors, devices, and systems. The Internet of Things (IoT) has provided, in recent years, the gradual exposure of society to these environments. However, Software Engineering requires specific techniques to deal with the development of these systems. Software Engineering must tackle the intrinsic characteristics of devices and sensors and complex interactions in intelligent environments to consolidate good development practices. **Objective**: The main objective of this article is to present a self-adaptive IoT architecture in an intelligent environment. The proposal concerns how different architecture modules cooperate and interact to develop new applications. **Method**: The work was developed through a real-world case study in an intelligent e-health environment. **Conclusion**: The results showed how a self-adaptive architecture using artificial intelligence can support the management of an intelligent e-health physical space. With this, it was possible to observe how data collection, environment monitoring, prediction of using IoT devices, and optimization of environment management can occur.

KEYWORDS

Self-adaptive, IoT, AI, e-health, Intelligent Environments

1 INTRODUCTION

The Internet of Things (IoT) has consolidated itself in people's daily lives and interactions with the environment in which they are [5]. Environments that use IoT characterize intelligent environments or smart ecosystems [28]. The IoT is responsible for allowing different devices and sensors to connect through the internet, allowing the sharing and collection of data about the environment [8].

The introduction of IoT brought several complexities to software development. Teams have to deal with some challenges [2], such as distributed devices and sensors, heterogeneous configuration of behavior, intrinsic characteristics, and integration and interoperability with other sensors and systems [22]. The systems interacting with these sensors and devices are critical parts of the intelligent environments in which they are involved [15]. Software must deal with the connectivity, management, scalability, and integration of sensors and devices in different contexts and quantities. Finally, developers are also affected, given the non-consolidation of good development practices, causing the arrival of low-quality systems on the market [20].

Therefore, the dynamics of intelligent environments are challenging topics for software development. With the expansion of IoT and intelligent environments, modern software engineering aims to support the development of systems for intelligent environments [20]. For this, software engineer experts study how different elements of these environments interact, behave, communicate, and cooperate, seeking to provide techniques and methods to help [14]. Examples of dealing with these challenges proposed by contemporary software engineering are IoT computational architectures, which define standards to be followed by developers to achieve quality attributes systems.

A complex, intelligent environment in this context is e-health. This environment is where interactions between sensors, devices, systems, and people can be essential to a person's life. These applications deal directly with other people's lives and how to facilitate and support everyday life [38]. In a complex and vital scenario, such failure to establish efficient computational architectures for these environments can lead to low-quality software, affecting society. IoT systems can use data generated by sensors and devices to adapt to situations occurring in the environment. This characteristic is related to self-adaptation, which may occur due to a user's need, the environment in which sensors and devices are found, or even to meet the security and functioning requirements of an environment. For example, a system that monitors patients in real-time when it detects an electric power outage cannot stop sending patient data, but it is necessary to adapt. Self-adaptive architectures are being used in other application domains. An example can be seen in [26].

Therefore, based on the challenges presented, the research problem addresses the construction of IoT architectures for intelligent environments, with the support of Artificial Intelligence (AI). The designed architecture aimed to adapt to the needs of an intelligent environment. The following research question was established to develop the architecture: "How have self-adaptive architectures been used to support intelligent environments?".

The use of self-adaptive architecture aims to support the different interactions in the complex and intelligent environment. In a scenario in which intelligent environments change people's daily lives, in addition to the rise of large-scale use of IoT devices and sensors, the need for architectural proposals for systems that deal with these complexities becomes even more evident. We developed

the work following the Design Science Research methodology. This methodology assumes artifacts (solutions) are designed to solve a practical problem [17]. A real case study was conducted in a corporate intellignet environment to evaluate the architecture. We developed a system using the architecture, demonstrating the interactions between people, devices, sensors, the environment, and the computational resources for each scenario. The case study results show how the use of a self-adaptive architecture supports the development of software for a complex, intelligent environment. As a contribution, we present the architecture to support the development of IoT applications that directly impacts people's daily lives.

The work is organized into six sections. The Section 2 deals with background knowledge. Section 3 discuss related works and an exploratory study. Section 4 detail methods and materials. Section 5 discuss a real case study. Finally, the last section presents the conclusion and future work.

2 BACKGROUND

Given the challenges of developing systems for intelligent environments, modern software engineering seeks to support technology teams that deliver software [22]. Gubbi et al. [13] show that one of the ways to support software development is through computational architectures. These architectures are intrinsic to all software, but they establish a set of characteristics that software must have to deal with most of the challenges and meet the interests of its end users. The authors also define that architectures must ensure that it is possible to capture, process and display data on a large scale; integrate and interoperability between devices, sensors, and systems; and scalability and flexibility for the entry of new devices. Furthermore, data security and privacy are topics of interest for these systems.

Self-adaptive software architecture solutions can be helpful for environments that have a changing nature and need different software configurations, as we can observe in intelligent environments. According to Weyns et al. [36], self-adaptive architectures must consist of two systems, a management system, and a managed system. The management system is responsible for adapting the managed system to the momentary demands of the software, for example, switching between a managed system module capable of doing more data collection and another capable of doing more data processing. The management system follows defined and mapped metrics to adapt the managed system to different configurations. These metrics may be related to response time, number of requests received, and number of failures. Managed systems are the systems that must handle the end use of the application.

Weyns et al. [36] define a self-adaptive system as one that can deal with changes and uncertainties in the environment, the system itself, and the objectives defined for it. These systems must also interact with the environment and have a feedback loop to collect information about what is happening with them. Finally, the authors state that self-adaptive systems have emerged as a fundamental element of modern software engineering.

Computational resources are essential for IoT. They provide the interface for devices and sensors to communicate with systems, allowing the collection and processing of data. Examples of what computational resources can help are related to edge, fog, and cloud processing, collecting data close to the generated source, and the cloud infrastructure needed to scale the number of devices and integrated sensors, ultimately storing the data for stakeholders to use. This stored data can still be processed and used for decision-making. Computational resources provide the computational structure for these processes [25].

3 RELATED WORKS AND EXPLORATORY STUDY

This section summarizes the main related works resulting from a systematic mapping of the literature on the research topic. All data relating to the systematic mapping can be seen in [12]. The systematic mapping was conducted to support our research conjectures (Section 4) and identify related works in the literature. We also present an exploratory study. The exploratory study reinforces our conjectures and presents the challenges and obstacles encountered in the area of interest.

Golec et al. [6] present a set of good development practices using serverless, focusing on security and privacy. Even presenting the use of serverless for the development of the study, the authors have privacy and security as their primary focus, not aiming to address the scalability and interoperability of IoT devices and sensors. Golec et al. also present an IoT application for e-health that monitors cardiac patients and helps in clinical decision-making. They used computational resources related to serverless connected to IoT devices to develop the system. Data processing and the analysis of the patient's situation use AI. Even though the work listed the importance of scalability of computational resources, the focus was on the application, the use of AI, and serverless rather than on how they developed the architecture. Also, the authors did not present aspects of how the architecture adapts to the environment's needs and the details about it.

Self-adaptive architectures for IoT applications are present in the literature. Alfonso et al. [1] present techniques to model and create self-adaptive IoT systems. The approach supported by the authors focuses on three main items: modeling interactions between devices, sensors, and computational resources; how to perform the division of services for deployment and how to ensure the architecture adaptation strategies. The authors present all the theoretical parts and how to implement the model. However, they do not validate the proposal. Furthermore, the framework must consider scalability and interoperability between sensors, devices, and computational resources. The solution proposed by the authors does not consider the need for self-adaptation according to the sensitivity in which the context is inserted. The authors focus on the adaptation strategy and not on how it is carried out according to the context.

Muthu et al. [23], Verma and Sood [35], Kaur, Kumar and Kumar [19], Verma, Agarwal and Gupta [34], and Tuli et al. [33] present the use of artificial intelligence based on data generated by sensors and devices, predict what types of diseases a person may be suffering or being affected by. Once again, AI is used in the final application, not alongside computational resources. Gupta, Al-Naime and Al-Anbuky [14] present the proposed architecture where artificial intelligence could be used. However, they do not present the use made in their work. AI is related to the application's data

processing, not computational resources. Talaat [31] proposes using artificial intelligence to predict real-time resource allocation. The work focuses on the algorithms that can support this prediction. Therefore, the author aims to verify which machine learning algorithms are best suited to reduce latency, improve service quality metrics, reduce response time, and ensure energy efficiency. The author focuses on demonstrating how algorithms work and not necessarily linking them to an architecture. Finally, the authors test the algorithms in a patient environment. Dai et al. [3] propose using AI to help predict and measure the energy expenditure that some aspects of the architecture can use. In this way, the authors hope there will be less energy expenditure. Dhasarathan et al. [4] propose using deep learning techniques to help predict privacy metrics in e-health environments. Deep learning techniques are applied at the application level rather than at the architectural level.

After reading and exploring the selected works, it was possible to observe that quality attributes are fundamental to the proposed architectures. Moreover, architectural proposals were identified for developing IoT applications in intelligent environments. The proposed architectures and their use demonstrate the impact of a good solution for creating quality and robust applications. Computational architectures are one of the fundamental factors for developing these applications with quality. Artificial intelligence is being used in some applications, such as predicting situations with patients, diseases, and scenarios. However, AI is not being applied in architecture, how quality attributes and computational resources behave.

Considering the research question, systematic mapping, and conjecture, an exploratory study was also conducted [10]. With the exploratory study, in addition to the difficulties and opportunities found in the literature, the aim was to observe other challenges that may arise during the development of IoT architecture environments and evaluate the use of computational resources. Furthermore, the objective was to validate and verify the challenges found in the literature in practice. Moreover, the execution of the exploratory study allowed us to learn more about the research question and our conjecture.

Therefore, we conducted two cycles of the Design Science Research methodology. The next section presents the Method and Materials, detailing the components of an architecture for intelligent environments that use IoT, exploring self-adaptation and AI to make resources available according to the demand of the environment in which the solution is inserted. The use of artificial intelligence will help to understand the dynamics that occur in the generation of data in the environment, providing learning when resources should be available and making them available through the computational infrastructure proposed in the architecture.

4 METHODS AND MATERIALS

Design Science Research (DSR) assumes that artifacts (solutions) are designed to solve a practical problem. Artifacts are objects, whose construction followed scientific methods in different steps to generate knowledge about a specific item or area of knowledge. The focus of the DSR is on the artifact and the relevance of its application [17]. According to Wieringa [37], research conducted using the DSR approach deals with two types of problems: practical and knowledge problems. Practical problems require changes in artifacts that are related to decision-makers. Knowledge problems portray necessary changes to existing knowledge about the world and the state of the art. Still, according to Wieringa [37], a practical problem is responsible for safeguarding the research; therefore, other practical problems and questions about existing knowledge will arise. Both problems and questions chain a cycle of knowledge construction.

To be able to perform the DSR execution, we establish the following conjecture. Suppose we propose a self-adaptive architecture for the development of IoT applications. In this case, can we improve the use of computational resources by systems, supporting data collection from intelligent environments and managing these spaces? As a theoretical basis, this conjecture derives from the knowledge we acquired in research we already carried out in [26], [10], [11] and is also presented in sections 2 and 3 of the present work through the presentation of concepts and the discussion of the solutions proposed by others in the literature. The following section presents the requirements established for implementing the architecture.

4.1 Requirements

The context is that of a corporate intelligent environment. It must be capable of dealing with a large amount of data generated by different sensors and IoT devices. Furthermore, the architecture must be able to deal with the available computing resources and use artificial intelligence to predict the use of computing resources. Finally, the architecture must self-adapt to the environment's needs.

The functional and non-functional requirements are established through the knowledge obtained in the systematic mapping and the exploratory study and the needs of an intelligent environment: FR01: The architecture must be capable of processing, storing, and loading data from different IoT devices and sensors in an intelligent environment; FR02: The architecture must be capable of allowing the integration of different databases, as well as the storage of this data; FR03: The architecture must display the stored data to support the management of the physical intelligent space; FR04: The architecture must adapt to the needs of the intelligent environment, enabling and disabling the use of computational resources; FR05: The architecture must use artificial intelligence to support computational resources.

As Non-functional requirements, we stated: NFR01: The architecture must be capable of dealing with different data models and databases, at least two, relational and non-relational models. Each IoT device and sensor has characteristics and data storage needs. Therefore, the solution must be flexible and extensible to address this characteristic; NFR02: The architecture must be capable of processing and storing large volumes of data. IoT devices and sensors generate large-scale data. For this reason, processing performance must be guaranteed; NFR03: The architecture must maintain low response time and adequate data processing. Some sensors and IoT devices can help with decision-making. For this reason, the performance and reliability of data processing can be crucial in an intelligent environment. Added to this is the need for some data processing to be carried out quickly; NFR04: Processed and collected data must be protected to avoid unauthorized access or leakage. Security and privacy must be guaranteed so that user and system data

are not exposed to undue people; NFR05: The architecture must be scalable to handle the increase in IoT devices and sensors. In addition to supporting the increase in the amount of data generated. Environments are changeable and can receive new sensors and IoT devices; for this reason, the architecture must be scalable; NFR06: The solution must be capable of using computational resources efficiently, providing their use only when necessary. The solution must adapt to the environment's needs to avoid wasting computational resources. In this work, efficiency is addressed by making computational resources available when necessary; NFR07: The solution developed via architecture must be easy to use. Monitoring and decision-making dashboards must be built in such a way that they can be used without problems. In other words, users must be able to visualize data and use decision-making tools without any problem; NFR08: The developed system must provide clear and intuitive views of what the data represents. Usability regarding applications that use data is essential when developing the architecture; NFR09: The solution must offer ways of accessing data for interoperability with other applications. Integrating data generated with other applications may be necessary as the architecture expands. In this way, the database must interoperate with other solutions.

4.2 First Cycle

Considering the conjecture, the research problem, the systematic mapping, the exploratory study, and the functional and non-functional requirements, the first cycle of the DSR methodology was executed. In the first cycle, an intelligent corporate environment was considered to develop an architecture. The proposed architecture was divided into modules related to different functional and non-functional requirements for its development. Figure 1 presents the proposed architecture.

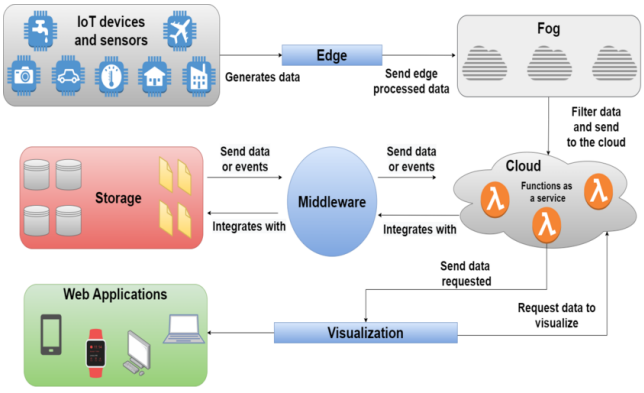

Figure 1: First IoT architecture developed

The architecture is divided into some modules, namely: **IoT devices and sensors**: Module responsible for all IoT devices and sensors in the intelligent environment. These sensors are responsible for generating data and can operate together or individually. Sensors and devices communicate directly with the edge infrastructure. **Edge**: Part of the architecture responsible for carrying out the first data processing using the capacity of devices on the edge. **Fog**: Part of the architecture responsible for continuing data processing after the Edge layer. Fog's infrastructure already has greater processing capacity. Edge and Fog allow greater data processing capacity and reduced response time, given their use close to the origin of data generation. **Cloud**: The cloud infrastructure used functions as a service, which used snippets of code responsible for dealing individually with each sensor and device. When a new sensor or device arrives, a new function must be built to be used as a necessary resource. At this point, the function transforms the data received by IoT sensors and devices into the data format required for storage. **Middleware**: Responsible for coordinating access to data, writing, and loading data servers. Furthermore, it connects functions as a service with the data layer. The use of middleware is necessary to guarantee security and data access policies and to allow coordination of functions as a service, ensuring that schemas are created for data consumption. These schemas deal with information about how functions can consume and access data. **Storage**: Responsible for storing data on file servers and databases. **Visualization**: Part of the architecture responsible for requesting and interpreting data that has been stored. This layer ensures that data is available for decision-making based on its display. **Web applications**: Various web applications for viewing data generated from the environment. These applications can be on cell phones, computers, and even on wearable devices (such as watches).

To deal with FR01, FR02, NFR01, NFR04, and NFR09, the architecture uses middleware. The middleware approach allows a function as a service for each sensor and device. Furthermore, the middleware has a layer that maps access to the database, generating access and control policies according to the needs of each application. Given that each device and sensor have a function as a service, data processing began to be distributed between different parts of the computing resources. This meant the load did not just remain in a monolith. Furthermore, using edge and fog allowed the filtering and processing of data close to the origin of its generation. This edge and fog approach also allows for reduced application response time. In this way, the needs of FR06, NFR02, and NFR03 were addressed. The architecture also deals with the proposition of a data visualization platform. For this, an application with dashboards can be specified. In this way, FR03, NFR07 and NFR08 were treated. Figure 2 presents the dashboard.

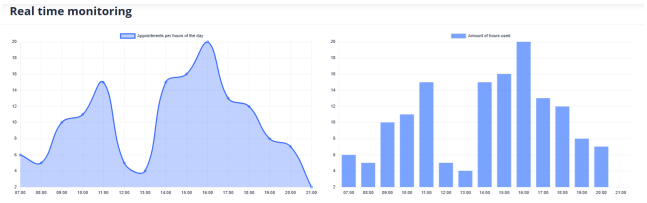

Figure 2: Developed application with dashboards

Although the first cycle of the architecture deals with several functional and non-functional requirements, some aspects can still need to be explored. They are:

(1) At certain times of the day, the computational resources are idle or receive a large load of data to process. The serverless

paradigm supported these characteristics, but the computational resources could be better used according to interactions in the scenario. An example can be seen in the intelligent an e-health environment operation; it operated between 6 am and 11 pm. However, all computational resources were available without real need throughout the day. Therefore, functional requirements FR04, FR05, and NFR06 were not addressed;

(2) The context in which the sensor is inserted directly influences the computational resources required. Various behaviors and actions can occur within an environment, for example, people walking, a sensor generating data, and a lack of electricity. In this way, the mapping of essential interactions to this environment must be privileged and mapped so that the computational resources are always prepared to receive this data. Functional requirements FR04, FR05, and NFR06 were not addressed as before;

(3) The usability of the dashboards and the tools to help the decision-makers could be improved, given the limitations encountered with the proposed visualization module. The NFR07 and NFR08 can still be better explored in a new architecture;

(4) The architecture presented how to deal with the entry of new IoT devices and sensors but did not present aspects related to artificial intelligence and the management of these computational resources. FR04, FR05, NFR05, and NFR06 were not treated.

Therefore, a second DSR cycle was conducted, to improve the architecture, considering the aspects previously identified and not explored in the first version.

4.3 Second Cycle

In addition to the aspects identified above, we considered the need of expansion of intelligent environments and their presence in people's lives [7]. The development of these applications requires flexible computational architectures that can handle the input of new IoT devices and sensors, scalability to ensure that it functions correctly in times of large-scale use and interoperability with other systems and devices [22]. Another need is supporting the interactions in these environments, adapting, and ensuring computational resource availability [22]. The use of serverless allows the creation of flexible computational applications adaptable to other cloud structures and can scale according to existing needs. In addition to these benefits, serverless allows computational resources to handle the most minor parts of an environment, such as sensors and IoT devices. The use of edge and fog allows data processing close to the origin of their generation. Allowing pre-processing and filtering of data to be performed.

We evolved the architecture developed in the first cycle to present a self-adaptive IoT architecture to deal with FR04, FR05, NFR06, NFR07, and NFR08. The self-adaptive IoT architecture provides a mechanism for choosing computational resources that meet the needs of intelligent environments – mapping the use of sensors and IoT devices and exemplifying how processes occur from data capture to use across the applications. The architecture is detailed in Figure 3, with emphasis on self-adaptive features.

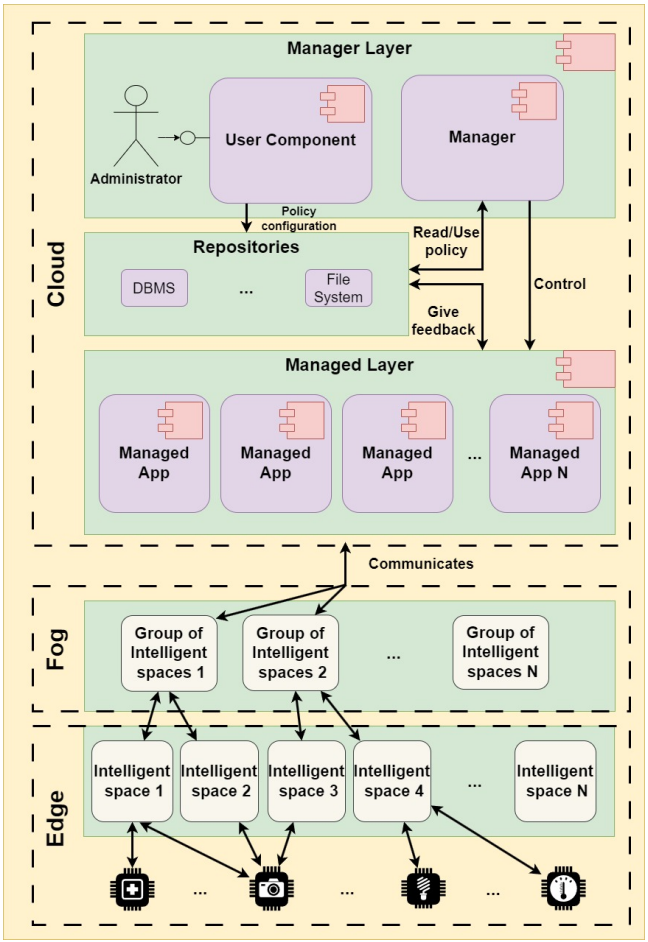

Figure 3: Self-adaptive Architecture proposed in the second DSR cycle

In the cloud, the first layer is called the **Manager Layer**. It is responsible for managing whether the Managed Layer meets the non-functional requirements established by the developers and stakeholders, such as scalability, interoperability, response time, memory usage, and processing usage. It is also responsible for changing the computational resources according to these definitions. The second component is the **Managed Layer**. It is the layer the final users use to monitor the intelligent environment. The Managed Layer will also be self-adapted according to the definitions established by the Manager Layer.

Connected to the **Manager Layer** and the **Managed Layer**, the repositories store data for the Manager Layer and also about the health of the Managed layer and the environment, such as response time metrics, amount of memory used, and amount of processing.

The self-adaptive architecture aims to adapt the use of computational resources according to the needs established through non-functional requirements and scenarios of using sensors and IoT devices.

An example of adaptation is that at a specific time of day, a computational resource is necessary for data processing, with up to

5 seconds of response time, from a temperature sensor to determine whether the environment is comfortable.

The Managed Layer is also responsible for running the application used by users. It has all the necessary computational infrastructure defined in the Manager Layer. It is composed of a serverless component that uses functions as a service to communicate with IoT device sensors; a middleware component to isolate access to the database from the application and allow the definition of usage rules; the database to store application data, the application component to follow the mapped intelligent environment; a workflow engine component for changing serverless component settings.

Applications are constructed to serve those interested in the data generated by IoT sensors and devices. In this way, the IoT systems in intelligent environments perform requests for middleware to obtain data. These functions communicate with the middleware and receive the requested data in return.

The last part of the Managed layer is the workflow engine. It is responsible for changing the configurations of computational resources according to the needs of the scenario. It executes the configurations defined by the Managed layer. The workflow engine has an API, and whenever it needs to change some computational resources, it sends an event to the cloud to execute the changes. More details about the workflow engine can be reached in [9].

One of the critical aspects is that one or more computational resources can interoperate, causing computational resources to run simultaneously. Furthermore, given the characteristics of the environment, computational resources can be scaled because of the loosely coupled cloud resources into the architecture. Each device and sensor act in this context, generating data related to its purposes. For example, a temperature sensor generates data about the environment. The data needs to be used by the managed system and end users. Computational resources are essential for this data's loading, processing, and visualization. In this article, our focus in on AI use. Therefore, the detailing of other modules can be consulted in [9].

4.4 AI processing

As previously stated, to adapt to the environment's needs, the architecture uses AI. In the context of intelligent environments, artificial intelligence is essential to optimizing the allocation of computational resources. By analyzing usage patterns, it decides when to activate or deactivate resources, dynamically adapting to the environment's needs. Furthermore, predictive analysis techniques predict future demand, reduce waste, and improve operational efficiency for more sustainable and economical management.

The proposed architecture uses artificial intelligence techniques. Examples of techniques that could be used in the architecture are machine learning, expert systems, recommendation systems, and automatic learning (AutoML). Some algorithms that can be used for these techniques are decision trees, random forests, linear regression, logistic regression, and collaborative filtering. One example is the use of AutoML. In the AutoML process, the decision on which algorithm to use is made through established criteria that evaluate the adequacy of the model to a given algorithm. The first phase of the selection process is preparing the test data. This phase executes the data cleaning and transformation. In the next phase, different algorithm models are evaluated on the data set. Each model's performance is evaluated using precision, recall, and F-1 score metrics. Given the models' performances, the worst are discarded. The models' hyperparameters are optimized to extract better performance from the models. Finally, in the last phase, there is the final choice of the model and which algorithm suits the chosen model [18].

Artificial intelligence trains with historically recorded data, allowing training sessions to be executed occasionally and allowing model learning and validations to be performed according to the time of year. One of the critical aspects is that one or more computational resources can interoperate, causing computational resources to run simultaneously. Furthermore, given the characteristics of the environment, computational resources can be scaled because of the loosely coupled cloud resources into the architecture.

In the proposal for using artificial intelligence, the characteristics of the model chosen via AutoML are taken into account when choosing an algorithm. This approach allows for greater assertiveness in using the algorithm with the model, allowing metrics related to the model to be optimized and explored. Given that the entire AutoML process checks the adequacy of the model to the algorithm, this use together brings benefits in terms of metrics related to precision, assertiveness and F1-score.

Artificial intelligence can help to improve the management of the intelligent environment, reduce the unnecessary use of computational resources, and, consequently, allow more efficient management of computational resources. Using loosely coupled cloud resources that can be easily activated and related to sensors is necessary for the architecture. The loosely coupled resources allow artificial intelligence to define which computational resource is vital for the scenario and understand when its use is necessary for the environment. Examples are machine learning, expert systems, recommendation systems, and automatic learning (AutoML).

For example, the use of AutoML allows the data generated from the application to be collected and pre-processed; algorithms can be tested, hyperparameters can be optimized, and training and evaluation of the models used.

The AutoML component is responsible for processing machine learning. This module trains and evaluates several machine learning models and selects the best model to be used for the data provided. This model is then used to analyze new data from resources, environment, and historical data to estimate the probability of occurrence of certain classes. A new predictive model can be selected and adjusted with each new input dataset or as demanded. This way, it is possible to maintain solutions generated by the architecture, for each specific application, about the predictive model used. After analysis by AutoML, the data has a label assigned by the selected AI algorithm and can be processed by expert and recommendation systems. The architecture uses historically recorded data, allowing training sessions to be executed occasionally and allowing model learning and validations to be performed according to the time of year.

The second cycle of the DSR aimed to deal with FR04, FR05, and NFR06. For this, the self-adaptive architecture was presented. The use of computational resources is presented according to the needs of the intelligent environment in which they are inserted. Furthermore, artificial intelligence was used to enable the processing

and self-adaptation of architecture to different scenarios in intelligent environments. The approach makes computational resources predictable, not allowing inappropriate use. This approach allows for reducing costs and waste of resources and a more adequate management of the environment's needs, whether or not they are available.

Therefore, the proposed architecture demonstrates the essential elements for developing IoT applications in an intelligent environment, focusing on computational resources using a self-adaptive architecture. The next section presents an initial evaluation of the architecture provided in this second DSR cycle.

5 EVALUATION

A real-world case study is detailed, to explore the research question: "How have self-adaptive architectures been used to support intelligent environments?". As previously presented, this study followed the DSR methodology. This section presents a specific case study. Other case studies can be accessed in [9]

5.1 Context

We designed the architectures and tested the application developed in a corporate e-health environment. The intelligent environment was a medical clinic with a coworking space, waiting room, and medical offices for patient care. We mapped the sensors and devices used in the medical clinic. Then, we talked with users of the intelligent environment to understand people's interactions, in addition to sensors and IoT devices. Finally, we identified the users of the environment.

The devices and sensors in the clinic spaces are: Coworking: Presence sensor, temperature sensor, and humidity sensor; Waiting room: Presence sensor, cameras, and totem; Doctor's office: tablet. The functioning of the presence sensor works to detect a person in the environment periodically. The presence sensor is available in all areas of the medical clinic. Temperature and humidity sensors, for example, perform coworking measurements every hour. The cameras are used in real- time to monitor the flow of people in the waiting areas. The totem is responsible for recording the arrival of patients for consultations and receiving payment for the services — the tablets in the doctors' offices record which health professionals are using the environment. As a result, all devices and sensors of the intelligent e-health environment were presented.

5.2 Preparation

Considering the sensors and IoT devices, as well as the environments present in the medical clinic, a mapping of the use of the environment was carried out and divided into two usage scenarios to be supported using the self-adaptive architecture.

- The first usage scenario ensures the professionals' coworking environment has a pleasant temperature. Temperature and humidity sensors generate data to support the management of this environment and ensure that it is comfortable. These sensors generate data about the environment every hour and notify the medical clinic coordinators.
- The second scenario deals with professionals' essential use of consultation offices to provide care to their patients. Whenever a health professional arrives at the clinic, he waits for his patient in the coworking space. The coworking presence sensor periodically sends whether an environment has people or not. Therefore, if a health professional arrives at the coworking space, the presence sensor sends an alert to the environment coordinators, warning them about using the area. When a patient is going to attend, the health professional leaves the coworking space and goes to the doctor's office designated for use. The presence sensor indicates the environment used when entering the doctor's office. Still, in the current scenario, a patient attending must notify the health professional of their arrival using the totem in the waiting room. The use of the totem alerts the health professional that the patient is already waiting in the room. A presence sensor in the waiting room identifies whether people are in that environment to support this alert. When the medical appointment finishes, a health professional clears the room, and the patient leaves. At that moment, the presence of sensors in the waiting room, and coworking space sent data on the release of the premises.

The data used in this phase of the architecture was generated in real-time. To this end, a one-week cycle was carried out to measure its operation not to harm employees' daily lives or the business environment's policies.

5.3 Implementation Strategies

For the implementation of the application, derived from the architecture, we used some specific technologies. Some are already implemented in the architecture.

OpenFaas [24] was used as the serverless technology approach. Given the need for IoT devices and sensors, in addition to the need for experimentation, functions as a service were written using Python [30]. The use of serverless with OpenFaas also made it possible to monitor the performance of functions as a service. Information such as response time, hour of use, number of requests, processing time, and memory usage are monitored and stored in a database. OpenFaas uses Prometheus [29] to manage and monitor performance metrics. OpenFaas allows the migration of the architecture's serverless structure to any cloud.

To use the Middleware layer, the architecture used Hasura [16]. Hasura offers integrations to different databases and allows the creation of data schemas. Different services, such as functions as a service, can consume their data and generate schemas.

We used a PostgreSQL relational [27] database to store scenario event data. The Thingsboard [32] was used to provide the displayed data, generate alerts, and display scenarios for those interested.

Figure 4 presents an example of a function as a service. The presented function as a service is a template for generating functions for new IoT devices and sensors. The exposed code points to loading, processing, and sending data for visualization.

OpenFaas uses functions as a service. Functions as a service offers flexibility to configure and allow each IoT device's and sensor's needs according to their characteristics. The serverless paradigm also allows for scalability using computational resources in daily situations that generate high demand. To verify the scalability of computational resources, the clinic's day-to-day experimentation

```python
import requests
import json

def handle(req):

    requestObject = json.loads(req)
    baseUrl = 'http://host.docker.internal:8081/api/v1/'
    key = requestObject['key']
    finalUrl = '/telemetry'
    url = baseUrl+key+finalUrl

    del requestObject['key']

    #Define custom logical here to the function as a service.

    requests.post(url, json = requestObject)
```

Figure 4: Custom function as a service code

allowed the visualization of scenarios with the need to expand computational resources and others where resources were not required.

OpenFaas allows the migration of the architecture's serverless structure to any cloud. This feature is essential to ensure computational resources' flexibility, scalability, and portability. To use the functions as a service in OpenFaas, it is enough to access the function's source code and create a YAML file. YAML files establish guidelines for using a computational resource, such as port, amount of memory, and processing capacity. Based on these settings in the YAML file, containers use functions as a service. Using containers allows the portability of functions and their activation and deactivation when necessary. In a scenario of different sensors and IoT devices, such as the clinic, the characteristics of each of the sensors can be privileged and met via the source code of the functions and through the configuration file that establishes the use of computational resources.

In addition to configuring computational resources by the management system, one of the critical factors is when we should have activated the computational resources. For this, data from the use of sensors and scenarios were essential. Based on the data stored from the execution of the scenarios in the Managed system, it was possible to predict the hours of the day that the computational resources were used. With this data, we developed an artificial intelligence model using Python's SciKit Learn [21]. The model used in the SciKit Learn library was the Linear Support Vector Classifier (Linear SVC). It is a linear classification model which uses a linear function for classifications. One of its characteristics is that it allows the separation of data according to their characteristics based on a linear hyperplane. Artificial intelligence aims to establish whether a computational resource should be available during a specific time of day. To train the artificial intelligence model to construct the model, we used historical data from the execution of sensors and IoT devices.

The API calls the self-adaptative executor if a computational resource must be available or unavailable according to the time of day. An autonomous agent generates an event by an API request. The API request generates an event to the OpenFaas to activate or deactivate a computational resource.

5.4 Conducting

Given that an intelligent environment composed of sensors and IoT devices, they are responsible for generating data about what is happening in the environment. The interactions between all actors form the intelligent environment. The temperature, humidity, presence, camera, tablet, and totem sensors use edge computing techniques to perform the first data processing. Edge usage is essential to help filter important data, assess the integrity of this data, and reduce the amount that must be stored. After processing the data initially at the edge, the data goes to the fog. The Fog structure is used in doctors' offices, waiting rooms, and medical coworking. Its use allows the data generated to be even more filtered, given the greater processing capacity of fog compared to the edge. By following the path of the data, after passing through the Fog structure, it will be processed in the Managed system. The Managed system has a Serverless, Middleware, Storage, and Application component. To verify the scalability of computational resources, the clinic's day-to-day procedures allowed the visualization of scenarios with the need to expand computational resources and others where resources were not required. Figures 5 and 6 present dashboards provided by the application.

Figure 5: Temperature/ Humidity control

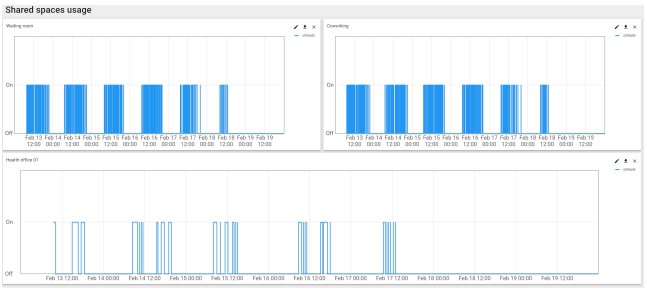

Figure 6: Dashboard presenting interactions

Executing the sensors and the proposed architecture in the corporate environment generated some data. A sample of this data, regarding execution over 15 days, can be seen on GitHub[1].

[1]https://github.com/mateusgon/architecture/

5.5 Analyzing

Through case study, the self-adaptive architecture was developed and evaluated to understand how much it supports data processing and management of the physical environment through computational resources. Artificial intelligence is also presented alongside computational resources, and its impact is discussed in the results subsection. Given the exposed scenario of interactions in the corporate e-health environment, the objective of the architecture was to support the use of computing resources in the intelligent e-health environment and improve the usability of applications.

In this way, the data generated by the intelligent corporate e-health environment was used, and the built application was evaluated. The results obtained through developing the architecture and monitoring of computational resources supported by artificial intelligence were satisfactory and are discussed in the following subsection.

5.6 Results

Implementing the architecture for the experimental scenario allowed us to obtain results and observations about the proposed architecture.

The scenario mapping was critical and conducted with professionals working in the e-health environment. These professionals helped to map when a computational resource was needed according to user interaction. Interactions by time of day were studied and passed to AI component based on historical data from the medical clinic.

Based on the information provided by professionals and historical usage bases, we also established weights for when a computational resource should be used. Therefore, at times of greater use of the environments, the chance of a computational resource being available was greater than at other times. These weights were passed on to AutoML training and helped predict whether a resource should be available or unavailable.

The main objective of the implemented application was to ensure that the necessary computational resources were adapted according to the time of day and the characteristics of an intelligent environment. In addition, the application should guarantee that it would be possible to address aspects of scalability, interoperability, and adequate access to the database, which are challenges addressed and resolved in other cycles.

One of the most significant challenges was the self-adaptation of computational resources to use sensors in clinical medicine. To make self-adaptation possible, we had to select and train a machine-learning technique with historical data from sensors. This training involved breaking down historical running data from sensors and devices by time and day of the week.

When evaluating whether a computational resource should be available or disabled, we found the challenge that the predicted artificial intelligence did not always have assertiveness. At times when the resource should be available, artificial intelligence predicts that it should not. While mispredicting usage causes some data not to be obtained, this has not impeded the testing of the intelligent e-health environment. Even though evaluating possible artificial intelligence or the one that would best fit was not the focus of our work, it was necessary to choose a model. For it, data was generated on its effectiveness. As a result of our experiment, we obtained some satisfactory data about the artificial intelligence model. We used as metrics, precision, recall, and F1-Score. Precision was about 70.9%, the recall was 61.1%, and F1-Score was 65.6%. As a significant benefit, this approach made it possible to guarantee support for the clinic's day-to-day activities. One of the ways to evaluate the impacts of predicting the use of computational resources was by talking to the workers who used our developed application. The reports of these workers showed that when there was no data, and data was necessary, the workers made themselves an inspection of the environment.

Three employees from the corporate environment were interviewed to evaluate the application developed. Two employees are clinic coordinators, and one of the interviewees is a manager. The interviews took place while the application was running in the corporate environment. The users reported that the data necessary for managing the intelligent environment, for the most part, were available for decision-making. Users also reported that when data was not available, manual verification of environments was required. Ultimately, the environment managers reported satisfactory experience with the developed application.

As main findings and observations, through the proposed architecture for developing IoT applications in e-health environments, we achieved the objective of improving the management of the use of computational resources and allowing data collection from the intelligent e-health environment and its management. This was demonstrated through the discussion, implementation, and results of the implementation of the artifact, as the results obtained are qualified as satisfactory. The self-adaptation of computational resources brought great benefits to the environment, allowing better management of resources, and ensuring that they will only be used when needed. This approach can allow the construction of applications to be increasingly focused on the complex interactions of existing environments and establish clear usage rules for computational resources. The use of the application also allows financial savings and even electrical consumption in case these computational resources are not used when not necessary. Therefore, the Research Question: "How have self-adaptive architectures been used to support intelligent environments?" could be answered. However, additional studies need to be conducted in similar environments with the aim of enriching scientific knowledge.

5.7 Threats to Validity

In this subsection, we discuss threats to validity that may limit or affect the results obtained.

Internal validation: During the case study, data was generated by specific sensors in the corporate environment. Although the initial results are valuable, a more detailed study in other intelligent corporate intelligent environments is necessary to identify other learnings and possible improvements. Adding new sensors and devices and competition for network resources can affect the data generated and obtained.

External validation: In a corporate scenario with an even more significant number of sensors, new techniques to predict the need to use computational resources will be necessary. The context, sensors, devices, number of rooms, and shared spaces in which the work

was developed will directly influence how computational resources should be used. New intelligent environments can generate different results using computational resources and data generation, bringing new learning and discoveries.

Construct validation: Different artificial intelligence models were not tested for computational resource selection during the case study. The use of other artificial intelligence can generate different results. Given the construction of the architecture and the division into modules, new artificial intelligence techniques can be used. New studies can be carried out to improve the results obtained with artificial intelligence, and researchers can compare results.

Validity of selection: During the study period, the selected corporate environment had similar people dynamics and use of computing resources. Using computational resources, people moving through the environment can generate new results on days with different behaviors. Therefore, validating the architecture on specific operating days may be necessary

6 FINAL REMARKS

The expansion of the Internet of Things and intelligent environments has led to new opportunities for Software Engineering. Intelligent environments address different areas of society that seek to automate or make life more accessible through technologies. Linked to the use of new technologies and the opportunities, challenges, and problems that have arisen for Software Engineering, the use of machine learning techniques and the development of computing resource technologies, such as cloud, edge, and fog, have allowed the development of a new range of applications.

As discussed, computational resources, large-scale data processing, quality attributes, and artificial intelligence could be used in IoT architectures in intelligent environments. Based on the systematic mapping and an exploratory study, the following research question was presented: "How have self-adaptive architectures been used to support intelligent environments?".

Considering the knowledge obtained in the systematic mapping and exploratory study, the work focused on the development of a solution to support intelligent environments. A self-adaptive architecture, using IoT devices, for intelligent environments were proposed to answer the research question and the established conjecture.

To evaluate the proposed architectures, using the DSR methodology, we used an intelligent corporate e-health environment. This environment had interaction dynamics between sensors, devices, people, and generated data. The exploration of this data generated through architecture allowed the development of an application made available to employees in an intelligent environment and supported them in day-to-day tasks.

As contributions to this work, we can cite:

- Development of architectures for the development of IoT applications. These architectures contribute to the scientific community through the knowledge generated and providing solutions that can be employed.
- Development of an artificial intelligence module to assist the use of computational resources by application architectures for IoT and intelligent environments. This contribution highlights this technology's advantages, problems, and challenges.

In future works, evaluating the architecture in environments other than e-health is essential to understanding adaptations that may be necessary. The main focus of the work was not on which artificial intelligence to choose but on validating the functioning of the architecture with artificial intelligence. Evaluating other artificial intelligence to predict situations with more scenarios and ensuring that resources are available in the most crucial moments is necessary.

Furthermore, other future work may address improving the accuracy of the artificial intelligence model and adopting new model training policies, exploration of other functional and non-functional requirements, improvements in the way non-functional requirements were handled, evaluation in other e-health environments, assessment in other intelligent environments.

7 DATA AVAILABILITY STATE

The datasets generated during and/or analyzed during the current study are available on [12].

ACKNOWLEDGMENTS

This work was partially funded by UFJF/Brazil, CAPES/Brazil, CNPq / Brazil (grant: 307194/2022-1), and FAPEMIG/Brazil (grant: APQ-02685-17), (grant: APQ-02194-18).

REFERENCES

[1] Iván Alfonso, Kelly Garcés, Harold Castro, and Jordi Cabot. 2021. Modeling self-adaptative IoT architectures. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 761–766.

[2] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. 2019. Software engineering for machine learning: A case study. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 291–300.

[3] Shijie Dai, Minghui Li Wang, Zhibin Gao, Lianfen Huang, Xiaojiang Du, and Mohsen Guizani. 2019. An adaptive computation offloading mechanism for mobile health applications. *IEEE Transactions on Vehicular Technology* 69, 1 (2019), 998–1007.

[4] Chandramohan Dhasarathan, M Shanmugam, Manish Kumar, Diwakar Tripathi, Shailesh Khapre, and Achyut Shankar. 2024. A nomadic multi-agent based privacy metrics for e-health care: a deep learning approach. *Multimedia Tools and Applications* 83, 3 (2024), 7249–7272.

[5] Lorik Fetahu, Arianit Maraj, and Abdullah Havolli. 2022. Internet of Things (IoT) benefits, future perspective, and implementation challenges. In *2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO)*. IEEE, 399–404.

[6] Muhammed Golec, Ridvan Ozturac, Zahra Pooranian, Sukhpal Singh Gill, and Rajkumar Buyya. 2021. IFaaSBus: A security-and privacy-based lightweight framework for serverless computing using IoT and machine learning. *IEEE Transactions on Industrial Informatics* 18, 5 (2021), 3522–3529.

[7] Jonas Gomes, Izaque Esteves, Valdemar Vicente Graciano Neto, José Maria N David, Regina Braga, Wagner Arbex, Mohamad Kassab, and Roberto Felício de Oliveira. 2023. A scientific software ecosystem architecture for the livestock domain. *Information and Software Technology* 160 (2023), 107240.

[8] Carles Gomez, Stefano Chessa, Anthony Fleury, George Roussos, and Davy Preuveneers. 2019. Internet of Things for enabling smart environments: A technology-centric perspective. *Journal of Ambient Intelligence and Smart Environments* 11, 1 (2019), 23–43.

[9] Mateus Gonçalo do Nascimento. 2024. *A self-adaptive IoT architecture to support computational resource allocation in an e-health environment.* Master's thesis. Computer Science Post-Graduate Program, Juiz de Fora, Brazil.

[10] Mateus Gonçalo do Nascimento, Regina MM Braga, José Maria N David, Mario Antonio Ribeiro Dantas, and Fernando AB Colugnati. 2021. Towards an IoT architecture to pervasive environments through design science. In *International*

Conference on Advanced Information Networking and Applications. Springer, 28–39.

[11] Mateus Gonçalo do Nascimento, José Maria N David, Mario Antonio Ribeiro Dantas, Regina Maria Maciel Braga Villela, Victor Ströele de Andrade Menezes, and Fernando Antonio Basille Colugnati. 2023. An Architecture to Support the Development of Collaborative Systems in IoT Context. In *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 1722–1727.

[12] Mateus Gonçalo do Nascimento, José Maria Nazar David, Mario Dantas, Regina Braga, and Victor Ströele. 2024. *Dataset: Computational resources in the development of e-health IoT applications: A systematic mapping study.* https://doi.org/10.5281/zenodo.10971320

[13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660.

[14] Akash Gupta, Khalid Al-Naime, and Adnan Al-Anbuky. 2021. Iot environment for monitoring human movements: Hip fracture rehabilitation case. In *Information and Communication Technologies for Ageing Well and e-Health: 6th International Conference, ICT4AWE 2020, Prague, Czech Republic, May 3–5, 2020, Revised Selected Papers 6*. Springer, 44–63.

[15] Yosra Hajjaji, Wadii Boulila, Imed Riadh Farah, Imed Romdhani, and Amir Hussain. 2021. Big data and IoT-based applications in smart environments: A systematic review. *Computer Science Review* 39 (2021), 100318.

[16] Hasura. [n. d.]. Hasura. Retrieved June 15, 2023 from https://hasura.io/

[17] Alan Hevner and Samir Chatterjee. 2010. Design science research in information systems. *Design research in information systems: theory and practice* (2010), 9–22.

[18] Shubhra Kanti Karmaker, Md Mahadi Hassan, Micah J Smith, Lei Xu, Chengxiang Zhai, and Kalyan Veeramachaneni. 2021. Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–36.

[19] Pavleen Kaur, Ravinder Kumar, and Munish Kumar. 2019. A healthcare monitoring system using random forest and internet of things (IoT). *Multimedia Tools and Applications* 78 (2019), 19905–19916.

[20] Xabier Larrucea, Annie Combelles, John Favaro, and Kunal Taneja. 2017. Software engineering for the internet of things. *IEEE Software* 34, 1 (2017), 24–28.

[21] Scikit Learn. [n. d.]. Scikit Learn. Retrieved June 15, 2023 from https:/scikit-learn.org/stable/

[22] Rebeca C Motta, Káthia M De Oliveira, and Guilherme H Travassos. 2018. On challenges in engineering IoT software systems. In *Proceedings of the XXXII Brazilian symposium on software engineering*. 42–51.

[23] BalaAnand Muthu, CB Sivaparthipan, Gunasekaran Manogaran, Revathi Sundarasekar, Seifedine Kadry, A Shanthini, and Antony Dasel. 2020. IOT based wearable sensor for diseases prediction and symptom analysis in healthcare sector. *Peer-to-peer networking and applications* 13, 6 (2020), 2123–2134.

[24] OpenFaas. [n. d.]. OpenFaaS. Retrieved June 15, 2023 from https://www.openfaas.com/

[25] Guadalupe Ortiz, Meftah Zouai, Okba Kazar, Alfonso Garcia de Prado, and Juan Boubeta-Puig. 2022. Atmosphere: Context and situational-aware collaborative IoT architecture for edge-fog-cloud computing. *Computer Standards & Interfaces* 79 (2022), 103550. https://doi.org/10.1016/j.csi.2021.103550

[26] Madalena Pereira Da Silva, Alexandre Leopoldo Gonçalves, and Mário Antônio Ribeiro Dantas. 2019. A conceptual model for quality of experience management to provide context-aware eHealth services. *Future Generation Computer Systems* 101 (2019), 1041–1061.

[27] Postgresql. [n. d.]. Postgresql. Retrieved June 15, 2023 from https://postgresql.org/

[28] Madhvi A Pradhan, Supriya Patankar, Akshay Shinde, Virendra Shivarkar, and Prashant Phadatare. 2017. IoT for smart city: Improvising smart environment. In *2017 international conference on energy, communication, data analytics and soft computing (ICECDS)*. IEEE, 2003–2006.

[29] Prometheus. [n. d.]. Prometheus. Retrieved June 15, 2023 from https://prometheus.io/

[30] Python. [n. d.]. Welcome to Python. Retrieved June 15, 2023 from https://www.python.org/

[31] Fatma M Talaat. 2022. Effective prediction and resource allocation method (EPRAM) in fog computing environment for smart healthcare system. *Multimedia Tools and Applications* 81, 6 (2022), 8235–8258.

[32] Thingsboard. [n. d.]. Thingsboard. Retrieved June 15, 2023 from https://thingsboard.io/

[33] Shreshth Tuli, Nipam Basumatary, Sukhpal Singh Gill, Mohsen Kahani, Rajesh Chand Arya, Gurpreet Singh Wander, and Rajkumar Buyya. 2020. HealthFog: An ensemble deep learning based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in integrated IoT and fog computing environments. *Future Generation Computer Systems* 104 (2020), 187–200.

[34] Ankit Verma, Gaurav Agarwal, and Amit Kumar Gupta. 2022. A novel generalized fuzzy intelligence-based ant lion optimization for internet of things based disease prediction and diagnosis. *Cluster Computing* 25, 5 (2022), 3283–3298.

[35] Prabal Verma and Sandeep K Sood. 2018. Cloud-centric IoT based disease diagnosis healthcare framework. *J. Parallel and Distrib. Comput.* 116 (2018), 27–38.

[36] Danny Weyns. 2020. *An introduction to self-adaptive systems: A contemporary software engineering perspective.* John Wiley & Sons.

[37] Roel Wieringa. 2009. Design science as nested problem solving. In *Proceedings of the 4th international conference on design science research in information systems and technology*. 1–12.

[38] Parang Zadtootaghaj, Ayoub Mohammadian, Bahareh Mahbanooei, and Rohollah Ghasemi. 2019. Internet of Things: A Survey for the Individuals' E-Health Applications. *Journal of Information Technology Management* 11, 1 (2019), 102–129.