informatics

MDPI

Article

A Self-Adaptive and Efficient Context-Aware Healthcare Model for COPD Diseases

Hamid Mcheick [1,*] and John Sayegh [2]

[1] Department of Computer Science and Mathematics, University of Quebec at Chicoutimi, Chicoutimi, QC G7H 2B, Canada
[2] Department of Computer Science, Faculty of Science Branch I, Lebanese University, Hadath, Beirut 6574, Lebanon; John.sayegh@st.ul.edu.lb
* Correspondence: Hamid_mcheick@uqac.ca; Tel.: +141-8545-5011-5676

Abstract: The emergence of pervasive computing technology has revolutionized all aspects of life and facilitated many everyday tasks. As the world fights the coronavirus pandemic, it is necessary to find new ways to use technology to fight diseases and reduce their economic burden. Distributed systems have demonstrated efficiency in the healthcare domain, not only by organizing and managing patient data but also by helping doctors and other medical experts to diagnose diseases and take measures to prevent the development of serious conditions. In the case of chronic diseases, telemonitoring systems provide a way to monitor patients' states and biomarkers in the course of their everyday routines. We developed a Chronical Obstructive Pulmonary Disease (COPD) healthcare system to protect patients against risk factors. However, each change in the patient context initiated the execution of the system's entire rule base, which diminished performance. In this article, we use separation of concerns to reduce the impact of contextual changes by dividing the context, rules and services into software modules (units). We combine healthcare telemonitoring with context awareness and self-adaptation to create an adaptive architecture model for COPD patients. The model's performance is validated using COPD data, demonstrating the efficiency of the separation of concerns and adaptation techniques in context-aware systems.

Keywords: software architecture; self-adaptation; context-aware system; COPD; separation of concerns; healthcare systems

Citation: Mcheick, H.; Sayegh, J. A Self-Adaptive and Efficient Context-Aware Healthcare Model for COPD Diseases. *Informatics* **2021**, *8*, 41. https://doi.org/10.3390/informatics8030041

Academic Editors: Kamran Sedig and Antony Bryant

Received: 8 May 2021
Accepted: 15 June 2021
Published: 22 June 2021

1. Introduction

Chronic obstructive pulmonary disease (COPD) has attracted research interest as a major public health problem. According to the World Health Organization [1], COPD is currently considered the fourth—and is positioned to become the third—most frequent cause of death worldwide [2]. It is also a disabling disease and is thus associated with high treatment and patient management costs. As the disease progresses, patients become more susceptible to respiratory exacerbations, which cause frequent hospital admissions and significantly impact patients' quality of life and healthcare costs [3,4].

Monitoring patients' health conditions from home or hospital and transmitting related data to a healthcare centre could be an excellent solution that facilitates the management of the growing number of COPD patients and reduces the burden on health services. This approach, called remote telemonitoring, can be used for timely assessment of an acute exacerbation or as a mechanism to generate alarms for patients and/or healthcare professionals when clinical changes occur that may constitute a risk to the patient [5].

There are many systematic reviews and studies on the topic of telemonitoring in respiratory patients, specifically COPD patients [6–9]. All of these studies have focused on proving the effectiveness of remote telemonitoring for COPD patients by studying the provided services and their impacts on the patient's quality of life, as well as the obtained organizational and clinical benefits. However, no one has yet proposed a comprehensive

and efficient telemonitoring system that helps control the burden of COPD. We aim to use telemonitoring to provide an application that helps COPD patients self-manage their disease and improve their quality of life, thereby reducing pressure on healthcare resources.

By definition, telemonitoring is the use of telecommunication technologies by patients for the timely transmission of data from home or hospital to a healthcare service centre [8]. Our objective was to develop a system that uses these data to provide effective interventions that prevent exacerbations through the early recognition of symptoms and prompt treatment, which may reduce the risk of hospitalization and control the burden of COPD.

In this telemonitoring system, we combine context awareness and self-adaptation with health telemonitoring, which enable our system to maintain awareness of the patient's data and context, adapt to relevant changes and act accordingly. These relevant changes are done based on the relevant parameters of COPD, identified in our work [10]. The adaptation process and the protective actions were extracted from a rule-based decision support system developed in three research papers [11–13]. However, each change in the patient context initiated the execution of the support system's entire rule base, which negatively affected its performance and involved unnecessary rules. We used the separation of concerns principle to divide the context and reduce the impact of changes in that context. In addition, we combined healthcare tele monitoring systems with context awareness and self-adaptation to provide an adaptive architecture model for COPD patients.

The remainder of this paper is structured as follows: Section 2 introduces the concept of context awareness and reviews the most common forms of self-adaptation frameworks. Section 3 highlights the characteristics and limits of self-adaptive systems. Section 4 describes our approach of combining techniques to build our self-adaptive healthcare system for COPD patients. Section 5 validates the proposed approach, and Section 6 presents a conclusion and proposes directions for future work.

2. Context Awareness and Self-Adaptive Systems

2.1. Background

In software systems, the notion of context awareness is generally coupled with the capacity for self-adaptation; otherwise, there is no reason to collect contextual data. Self-adaptation is the process of reorganizing, restructuring and reconfiguring a system in response to changes in resources or the system environment [14]. The emergence of complex and pervasive information systems, smart systems and the Internet of Things has made the concept of self-adaptation a point of interest for both researchers and engineers, leading to the introduction of a new category of systems called self-adaptive systems.

Self-adaptive software modifies its own behaviour in response to changes in its operating environment. By "operating environment," we mean anything observable by the software system, such as end user input, external hardware devices and sensors, or program instrumentation [15].

2.2. Self-Adaptation Frameworks

A framework manages the lifecycle of system components and separates the different layers based on the model–view–controller design pattern. In addition, it offers a configuration mechanism and various services, such as logs and security. This section provides an overview of several self-adaptation frameworks that have been discussed in the literature and identifies their characteristics and limitations.

2.2.1. FUSION

FeatUre-oriented Self-adaptatION (FUSION) is an adaptation framework to unanticipated conditions and efficient run-time analysis [16]. Unlike other frameworks, it does not provide an analytic model that designers must consider in the design phase. This problem is resolved because the system is required to achieve a number of quality-of-service (QoS) goals, such as security and performance, and provides a solution for each QoS concern. For example, to achieve good performance, it provides caching, and for security, it provides

authentication. As the system changes, it enables or disables features in order to adapt to a certain goal (e.g., response time). After analyzing the concept, the framework is built based on two core principles: features and goals. A feature is independent of a particular system implementation or application domain. It abstracts the capabilities of a system (i.e., what it will do) and may affect functional (e.g., search for a product) or non-functional (e.g., authentication) properties. The goal represents the QoS objectives for a particular scenario.

2.2.2. CASA

The Contract-based Adaptive Software Application (CASA) framework [17] is specialized in handling resources instability. The framework presumes that a system should not make any assumptions about the resources that will be available and should be prepared for any resource availability scenarios. The application should continue to operate at different levels of performance or functionality. To achieve this, CASA involves two main components:

1. the CASA framework, an integrated framework to develop adaptive applications;
2. CASA runtime, which provides resource awareness and dynamic adaptability to applications in a transparent way.

The framework provides an integrated approach to include all kinds of service parameters across different application domains within the same framework. An adaptive application resides on distributed autonomous nodes that form ad hoc networks. At runtime, when a peer application decides to interact to negotiate a service agreement, the CASA runtime system (to satisfy the service commitments of individual applications) uses proper resource allocation and management techniques. If there is a collision—that is, nodes that want to use resources at the same time—CASA carries out dynamic reconfiguration of the application components.

2.2.3. SSOA

The Specific to Service-Oriented Architectures (SSOA) framework [18] specifies any kind of adaptation by performing a precise decomposition of the different functionalities. Each of these functionalities may be specialized to fit a particular need. The framework can then provide clear, well-structured code that is easy to maintain and can evolve for new use cases. SSOA provides five major possible adaptations:

1. Parametric adaptation, which modifies the value of an existing parameter or service;
2. Functional adaptation, which replaces one function implementation with another, leaving the interfaces unchanged;
3. Behavioural adaptation, which changes how a service acts and possibly its interface;
4. Structural adaptation, which modifies the composition of services inside an application;
5. Environmental adaptation, which allows the outside world of the application to be changed (service migration).

The SSOA framework works at different levels:

1. Single service;
2. Composition of services in one application;
3. Several applications running on heterogeneous service-oriented platforms.

2.2.4. CAreDroid

Mobile applications adapt to the proximity of users or devices, changing locations, connectivity states and available resources. Due to the lack of support for context-aware applications in mobile systems, developers must build their own engines to support context awareness. The engines that they build handle specific sets of physical conditions. Therefore, the application may ultimately be unclear and difficult to maintain because of the separation between functional and non-functional code.

The Context-Aware for Android (CAreDroid) framework [19] aims to separate these two parts. It allows developers to focus only on the logical business of an application by

providing a list of methods for certain contexts, while leaving complex decisions to the framework. The framework monitors the contexts at runtime and activates methods only when it intercepts calls to sensitive methods. CAreDroid is implemented as part of the Android runtime system and has two main advantages. First, it increases the efficiency of applications, and second, it simplifies them by offering automatic and dynamic services. This is in contrast to applications that use only the standard Android API. To use CAreDroid, the developer must specify what the application should do in a certain context. In addition, the framework automatically handles methods, and the mapping of these methods to contexts is defined in the configuration file. This has the further advantage of reducing how much code the developer needs to write. The framework solves the indirection problem of reading sensor data at the application layer by introducing context monitoring at the system level.

2.2.5. Self-OSGi

Built on Java technology based on the OSGi framework, the Self-OSGi framework [20] proposes the adoption of the belief–desire–intention (BDI) agent model. Its components and service-based software systems can be built over this BDI model, which has self-properties. It resolves the adaptation problem using component-based and agent-oriented software engineering, both of which offer modular design that allows different system functionalities to be encapsulated, integrated and organized. The focus of Self-OSGI is the unification of agent, components and services. At its core, it deals with BDI, component containers and components services adaptation.

BDI is based on three models:

1. The belief model, which describes information about the environment and internal state;
2. The goal model, which describes the goals an agent may have and how these goals can be achieved;
3. The plan model, which describes the plan available to the agent for the purpose of achieving its goals.

A component container can be viewed as a wrapper that deals with technical aspects such as synchronization, security and persistence. Component containers must have a technical interface so that all components can have a uniform means of accessing the services provided. The separation of components' services and how they are implemented is key to building self-adaptive architecture with Self-OSGI. A formal base is required to describe the provided and required features of individual components and semantic aspects.

2.2.6. Dynamic Pervasive Healthcare System for COPD

This system proposes a new vision of telemedicine and remote care solution that promotes self-management and self-adaptation for COPD patients using advanced decision-making technique [12]. The findings of this system showed that dynamic thresholds can enhance existing tele monitoring systems and therefore help to identify the health status of COPD patients. However, this system is more complex and less efficient than the traditional systems because it must deal with complete rules (more than 20 thousand rules), large ontologies and relational databases simultaneously. The concept of integrated COPD care services is still in its infancy [12,21].

Other research projects have been recently developed for COPD disease. Humphries et al. [22] proposed a solution based on DL to enable the automatic classification of emphysema patterns at CT. A DL algorithm using a convolutional NN and long short-term memory architecture was trained to classify the pattern of emphysema according to Fleischner criteria in a retrospective analysis of the genetic epidemiology of COPD (COPDGene) study [23]. These classification projects did not use a context and dynamic classification approach and reasoning system.

2.3. Limitations of Adaptation Frameworks

The main limitations of the previous frameworks are as follows: They do not provide a comprehensive mechanism by which to reflect the context in the system or to separate the adaptation logic, although some frameworks have introduced the parameters approach that we use in our system. In addition, those frameworks introduced this approach as an abstract concept without providing any details regarding how to manage system parameters or use them with a rules engine to activate the adaptation process. Especially, the system developed by Ajami et al. [12] does not separate the rules into different blocks (units) to avoid their execution for each context change.

3. System Requirements and Self-Adaptation Characteristics and Taxonomy

Based on previous studies of frameworks and architectures, we identified the main criteria and requirements that characterize self-adaptive systems in order to develop our self-adaptive system's architecture and form our approach.

3.1. Requirements Extraction and Gathering

To design a self-adaptive system, the first step is to accurately identify the system requirements. Here, we used the W5H pattern (Where, When, What, Why, Who, How) [24], which presents six questions intended to elicit adaptation requirements:

- Where: Determines where the change needs to be implemented;
- When: Addresses temporal aspects of change;
- What: Identifies what attributes or artifacts of the system can be changed through adaptation actions and what needs to be changed in each situation;
- Why: Deals with the motivations for building a self-adaptive software application;
- Who: Addresses the level of automation and human involvement in self-adaptive software;
- How: Determines how adaptable artifacts can be changed and which adaptation action(s) are appropriate for application in a given condition.

Next, we answered these questions individually to obtain an overview of our system's requirements (Table 1).

After identifying our system requirements, we obtained an overview of self-adaptation perspectives and aspects in order to understand the criteria that could help us design our system architecture.

3.2. Adaptation Characteristics and Taxonomy

Krupitzer et al. [14] presented a taxonomy of the various properties of self-adaptive software. We analyzed their work, projected this taxonomy onto our system requirements and used the results to build our system. These issues must be addressed after the requirements gathering phase in order to properly design the architecture.

3.2.1. Time

This dimension was identified through the "when" question: when do we need to adapt? Handte et al. [25] provided two perspectives on the temporal aspects of adaptation: (i) reactive and (ii) proactive. Reactive adaptation refers to adaptations that occur whenever there is a change in the context and the monitored data are analyzed for abnormal patterns. In proactive adaptation, the monitored data are used to forecast system behaviour or environmental state [14]. In our case, we needed to combine both reactive and proactive adaptation, returning to the model presented by Ajami and Mcheick [11] (Figure 1). The adaptation occurs on the user and physician sides and is reactive to changes in user contextual data (vital signs, environmental risk factors and planned activities). For example, the system sends an alarm to the user or physician about dangerous exacerbation or automatically books an urgent medical appointment. Proactive adaptation is used to extract new rules from both (i) monitored data and (ii) treatment evaluation, and the rule-based reasoning engine can be updated with these extracted rules.

Table 1. Requirements Extraction.

Context Requirement	Description
Where	Where do we need to make a change inside our system when a context changes? In the model presented by Ajami and Mcheick [11], change needs to take place in the application layer on both sides: user interface and physician interface.
When	When do we need to make these changes? Changes should take place whenever an urgent update occurs in user contextual data, such as vital signs, environmental risk factors and planned activities, or periodical changes, such as treatment evaluation and decision support suggestions.
What	What do we need to change? We need to update some system attributes that present the system state. These attributes could in turn trigger new functions or activate new components.
Why	Why are these changes required? In healthcare monitoring applications, especially these related to chronic diseases, taking preventive action is a crucial component of treatment plans. In addition, the ability to notify the patient and medical experts about any threatening situation or abnormal signs makes these applications more efficient.
Who	Is any human intervention required in the adaptation process? On the patient side, all biomedical data and surrounding environmental data are collected from sensors. However, because physical activities affect COPD patients' states, patients need to be able to specify their planned physical activity (e.g., running, swimming) and the system needs to be able to detect these activities.
How	How should we determine what changes and actions are needed in the adaptation process? Ajami and Mcheick [11] provided a rule-based reasoning engine. All required actions and changes can be deduced based on these generated rules.

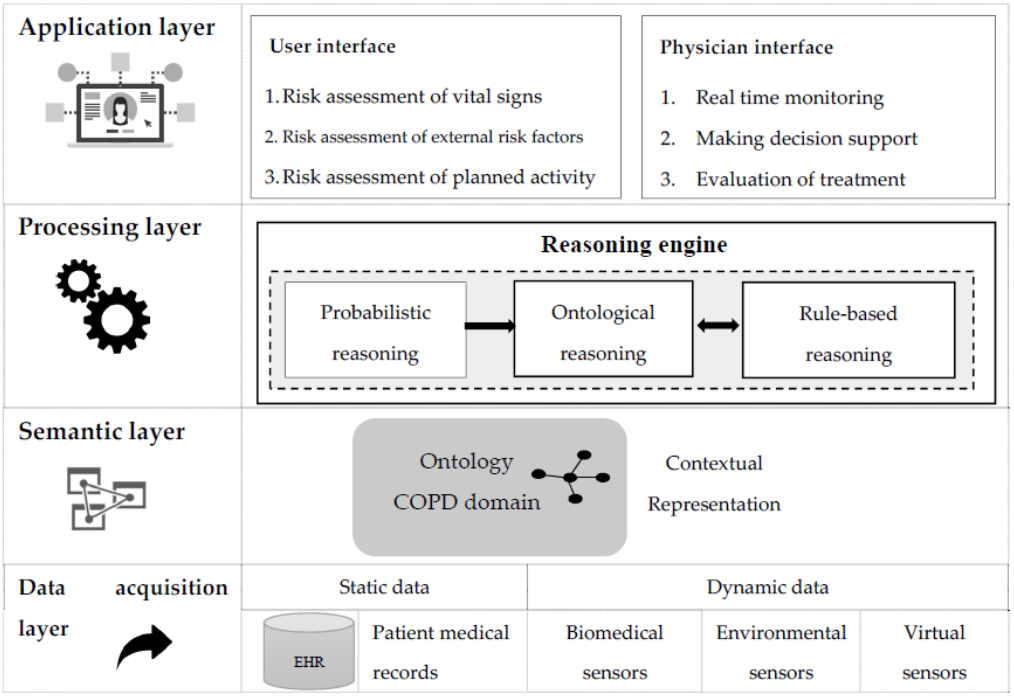


Figure 1. Framework of our COPD decision support system [11].

3.2.2. Reason

This aspect is related to the "why" question. Building a self-adaptive system is more complex and expensive than building a conventional system. Therefore, when we decided to develop a self-adaptive system, it was necessary to provide a convincing answer to the question of why the adaption of context is needed. Adaptations may be triggered for three reasons: (i) changes in context, (ii) changes in technical resources and (iii) changes in the user input.

In our case, adaptation is triggered due to changes in context. This offers a potential solution to the multiscale nature of COPD, where we need to detect all external and environmental irritants for each COPD patient based on his or her profile data and adapt the safe range for all biomarkers to detect any indicator of an exacerbation. In addition, we need to adapt the suitable range of the surrounding environmental conditions to prevent any potential threat to patient health. Note that some context changes should be inserted manually, such as planned activities.

3.2.3. Level

At which level do we need to apply our changes? To answer this question, we needed to be aware of the different levels of our system. The levels where the adaptation could be applied are shown in the taxonomy in [26]. In light of our system requirements and framework model (Figure 1), the changes in our system are handled in the application and processing layers. In the application layer, we need to update the acceptable range for the various datasets (e.g., vital signs, temperature, humidity, acceptable physical activities), activate new components or call new functions (e.g., alarming systems, providing medical advice, offering recommendations). In the processing layer, which contains the reasoning engine, the change is limited to the process of updating the rule base with the new extracted rules.

3.2.4. Technique

What kind of change is needed? McKinley et al. [27] provided two approaches for adaptive software: (i) parameter adaptation and (ii) compositional adaptation. Parameter adaptation modifies system behaviour by adjusting system parameters, whereas compositional adaptation enables the dynamic exchange of algorithms or system components at runtime. Our system uses the first approach, as it is suitable for a rule-based system where we can update system parameters depending on the rules. Even rules that are dependent on each other can be mapped to different parameters that are likewise dependent on each other. We used this approach to apply all the actions specified in our system, which we describe in Section 4.

3.2.5. Adaptation Control

A self-adaptive system consists of the adaptation logic and the adapted resources. Two approaches for implementing the adaptation logic can be found in the literature. The internal approach adjusts the adaptation logic based on the system resources, while the external approach splits the system into adaptation logic and managed resources. The IBM Autonomic Computing Initiative provided an external, feedback-control approach called the Monitor-Analyze-Plan-Execute (MAPE) model [28]. The MAPE loop highlights four essential aspects of self-adaptation:

1. Monitor: The monitoring phase extracts information—i.e., properties or states—from the managed element.
2. Analyze: This phase determines whether something has gone wrong in the system, usually because a system property exhibited a value outside of expected bounds or has a degrading trend.
3. Plan: This stage determines a set of actions to adapt the managed element when a problem is detected.
4. Execute: This phase applies a chosen set of actions to effect changes in the system.

Another aspect of adaptation logic is the degree of decentralization. Centralized adaptation logic can be a solution for systems that have few resources to manage. Since our system is a large telemonitoring system with many components to manage, we followed a decentralized approach, implementing independent units that controlled different aspects of adaptation. Applying the MAPE model [28] divided into different adaptation units ensures maintainability, scalability and adherence to the separation of concerns principle.

The next section describes how we combined the above requirements, characteristics and criteria to build our adaptation system, which we propose integrating with the solutions previously provided in "Ontology-Based Model to Support Ubiquitous Healthcare Systems for COPD Patients" [11], "A Pervasive Healthcare System for COPD Patients" [12] and "Ubiquitous Healthcare Systems and Medical Rules in COPD Domain" [13].

4. Self-Adaptive Healthcare System for COPD

4.1. Pervasive Healthcare System for COPD Patients

Ajami and Mcheick [11] designed and validated an ontology-based approach to keep track of patients' physical status, suggest recommendations and deliver interventions in a timely manner. Their decision support system created safe environments for COPD patients based on an ontological formal description of a health-related domain that used Semantic Web Rule Language (SWRL) rules. The SWRL rules of their system were constructed from medical guidelines, research and independent expert opinions to estimate the risk of COPD exacerbation. This work was expanded in [12,13] by Ajami et al., who proposed a specific domain architecture for COPD by providing an intelligent monitoring infrastructure guided by rules. This architecture consists of the following four layers:

1. In the acquisition layer, different sorts of data—such as the medical profiles of COPD patients, biomarkers and environmental information—are collected and transmitted from monitoring sensors and wearable devices.
2. The semantic layer translates the real context of the patient into machine-understandable and accessible language.
3. The processing layer aims to detect all possible hazardous events that could influence the COPD patient. In the healthcare system, we provided rules expressed in SWRL to describe all implications and consequences. These rules are used by an inference engine to derive new facts, detect events and predict potential risk factors.
4. The application layer is divided into two parts (a patient interface and a physician interface). Each part provides a set of functionalities and services related to the patient health state, risk assessment for vital signs and external risk factors, and many more.

The first goal of this paper is to improve the efficiency of existing COPD systems. The second goal is to provide an architectural design for the application layer that addresses the connection between three parts of the general architecture (Figure 2). These parts are as follows:

1. The end user application should provide needed services for both patient and physician, such as:
    - Alerting both the patient and physician to any exacerbation of vital signs (body temperature, blood pressure, heart rate, partial pressure oxygen [PaO_2], oxygen saturation [SpO_2], partial pressure carbon dioxide [$PaCO_2$], oxygen consumption [VO_2], respiration rate, blood pH, bicarbonate [HCO_3] and FEV1);
    - Providing recommendations and alerts to the patient related to conditions in the external surrounding environment and the patient's physical activities (e.g., warning the patient when the weather temperature and humidity exceed a certain range for a long duration; suggesting that the patient stop a certain physical activity, like walking, when their biomarkers indicate an abnormal situation);
    - Continuously adapting the acceptable ranges for patient biomarkers and external conditions;

- When a grave situation is detected, alerting emergency personnel or informing both the patient and physician about the need for an urgent appointment.

2. The data sources (sensors and patient records) provide a continuous stream of contextual data and historical data about the patient.
3. The rule base serves as the knowledge base for our system. These rules (20,308 rules) are generated and identified based on the literature of COPD diseases and validated by 20 physicians to verify the profile of patients, detect their location, evaluate the status of patients and offer the recommendation services [12].

4.2. Adaptation Approach

To build a self-adaptive system, we analyzed the adaptation approaches in the literature by studying many frameworks. To build these systems, we distinguish two approaches: parameter adaptation or compositional adaptation. Compositional adaptation enables the dynamic exchange of algorithms or system components at runtime and improves performance by adding new components or adjusting the system in response to new circumstances [27]. Parameter adaptation modifies system behaviour by adjusting system parameters. This can generally be achieved quite easily, as the adaptation logic must only control and change parameters. However, changing parameters can involve high complexity if the parameters are dependent on each other [27].

Parameter adaptation is well suited for rule-based systems and for reflecting the context in a system. Accordingly, we adopted this approach in our research. However, we needed to address the complexity of changing parameters (analyzed in the next section).

4.3. Adaptation Life Cycle

Since our system was developed to monitor the user's context, it was necessary to find the simplest way to handle recurrent changes in the data and the adaptation actions required by these changes. The main challenge was to form the pattern of the data flow in the system and identify how system functionalities should be triggered depending on the context.

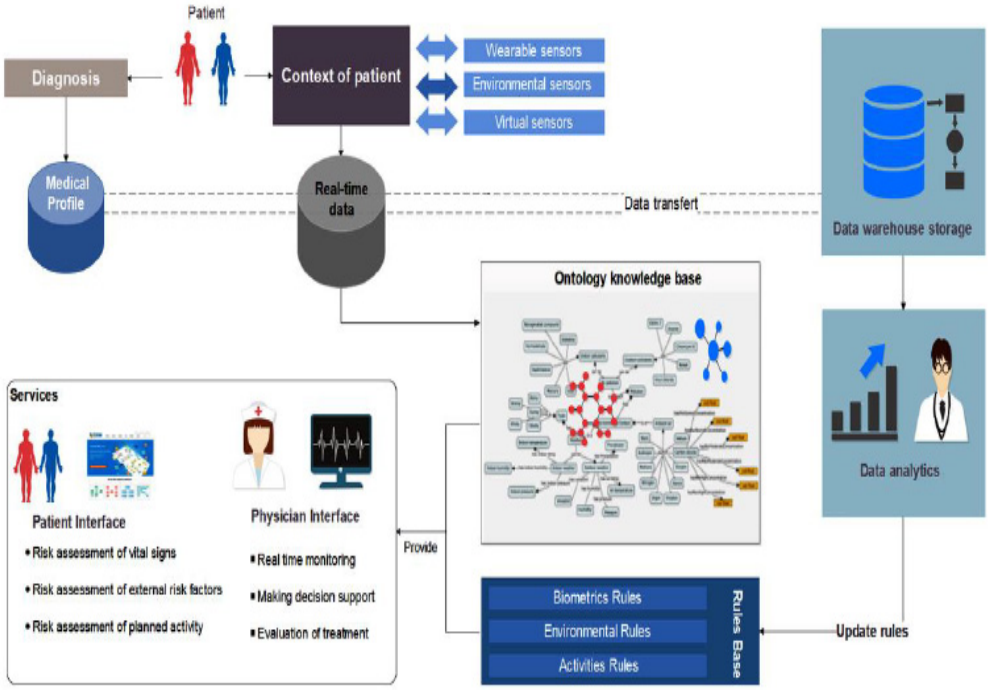


Figure 2. General architecture for ubiquitous COPD healthcare systems [13].

The lifecycle of our system is pictured in Figure 3, which shows how the MAPE model is applied in our system. When a context is changed, the Monitor unit obtains

attributed values, then the Analyse unit detects abnormal situation. The Unit Plan triggers functionalities to execute rules and adapt the system based of these attributes and finally, the Unit executes the actions to provide the services for patients.

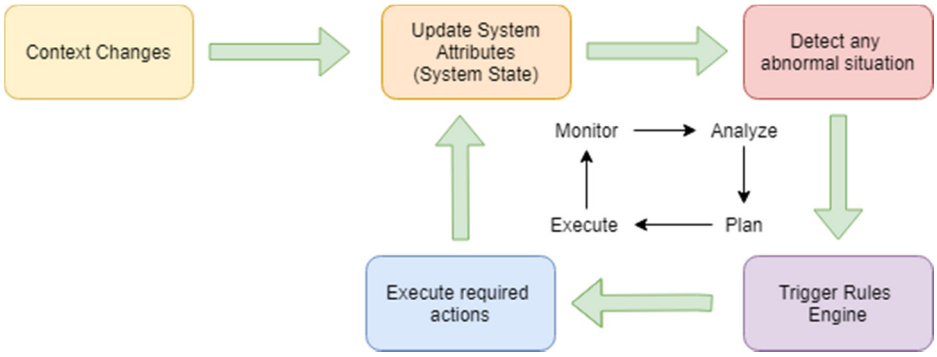


Figure 3. Adaptation lifecycle.

Whenever a contextual change occurs, it is directly reflected in the system state, which contains all the contextual values being monitored. These updates in turn trigger the responsible adaptation unit to detect any abnormal situation and send the relevant data to the rules engine to decide what action should be taken. Once the adaptation action is determined, the system attributes are updated in order to activate the required service.

As mentioned before, we applied the parameter adaptation technique, meaning that not only contextual and patient data are stored in the system state but also system attributes. These attributes are responsible for the adaptation process and for triggering the system's functionalities.

To manage the system state, we followed a pattern called unidirectional data flow, a mechanism used in web-based application frameworks like React and Redux [29]. A unidirectional data flow means that all data in a system follow the same lifecycle pattern, making the logic of the system more predictable and easier to understand. It also encourages data normalization, thus avoiding the creation of multiple independent copies of the same data that are unaware of one another. This is achieved by imposing certain restrictions on how and when updates can occur. These restrictions are reflected in the following three principles:

1. Single source of truth: The system state is the only place in the system where the data are stored, and any system component that needs to read those data or receive the latest updates must be listening to it. In addition, any update in the data is directly reflected in the state.
2. One-way data flow: There is one and only one way for data to be transferred to other parts of the system. This means that system units cannot directly send changes to each other. In addition, one system component cannot directly call any function or modify any variable in another component.
3. One way to change the state: The data stored in the system state can be changed only by using predefined functions called "actions." Therefore, no individual system component can immediately update the state. For example, if it were necessary to update the safe range for outdoor temperature, the system would need to use the predefined action "updateOutdoorTemperatureSafeRange".

4.4. Decentralization and Separation of Concerns

As mentioned in the previous section, the degree of centralization is one of the main criteria to consider when building a self-adaptive system.

Our system is a large system with many components to manage and a huge amount of contextual data to monitor and connect to the rule engine (which consists of 20,308 rules). We therefore followed a decentralized approach by implementing independent units

that control different aspects of adaptation. Applying the MAPE model divided into different adaptation units enhances performance and ensures maintainability, scalability and separation of concerns.

Our system is based on the following three main components:

- Multiple adaptation units: Each unit is responsible for handling the monitoring process for a specific type of contextual data and updating a set of system parameters associated with the same type of monitored data.
- Shared memory unit: This unit, called the global state, consists of multiple substates, each of which saves system parameters, such as environmental and biometrics safe ranges, as well as functional parameters, which are responsible for triggering services.
- Central adaptation unit: This unit is responsible for monitoring the updated system parameters saved in the global state and triggering the required services accordingly.

4.5. Adaptation Engine and Monitoring Units

Our main contribution is given in processing layer by designing many units and adaptation engine (Figure 4). The adaptation engine consists of a central adaptation unit and multiple adaptation subunits. Each subunit is responsible for monitoring and managing changes for a specific category tuple (data, rules and services).

The system variables are saved in a shared memory called the global state, which is a composition of substates. Each substate is considered a container for saving category-specific data and is updated and managed by the adaptation subunit that is responsible for monitoring the same category.

We divided our sets of data, rules and services into five categories (modules or units), given in Table 2. This categorization was based on the categories of the context parameters, which include biometrics and environmental parameters, and the factors affecting the patient, which include user activities, user location and the duration of each parameter.

Each subunit is responsible for a specific category of tuple. Five subunits were identified (Figure 5):

1. Biometrics unit;
2. Environmental unit;
3. Activities unit;
4. Location unit;
5. Duration unit (time unit).

The first monitoring unit is the biometrics unit, which is responsible for monitoring all biomarkers coming from patients' biometrics sensors and reading all the data stored in the other substates. Depending on the related rules in the rules engine, it then updates the biometrics state with the safe ranges for all vital biomarkers and their current measurements.

The second monitoring unit is the environmental unit. Its responsibility is to monitor all the streamed data from the environmental sensors (e.g., weather temperature, humidity, air pollution percentage), read the stored values in the other states and update the environmental state with the safe range for each environmental and external factor, depending on the environment-related rules.

The third monitoring unit is the activities unit. It is responsible for detecting the patient's daily routine, current physical activity and plans. Detection can be performed either through an explicit request from the patient app to fill out a daily schedule with a predefined set of activities (working, sleeping, eating, exercising, running, walking, etc.) or by using sensors (motion detectors) to detect what physical activity the patient is performing. This unit also reads the stored data in the other substates and updates the activities state with the set of allowed activities and current and planned activities, depending on the activity-related rules.

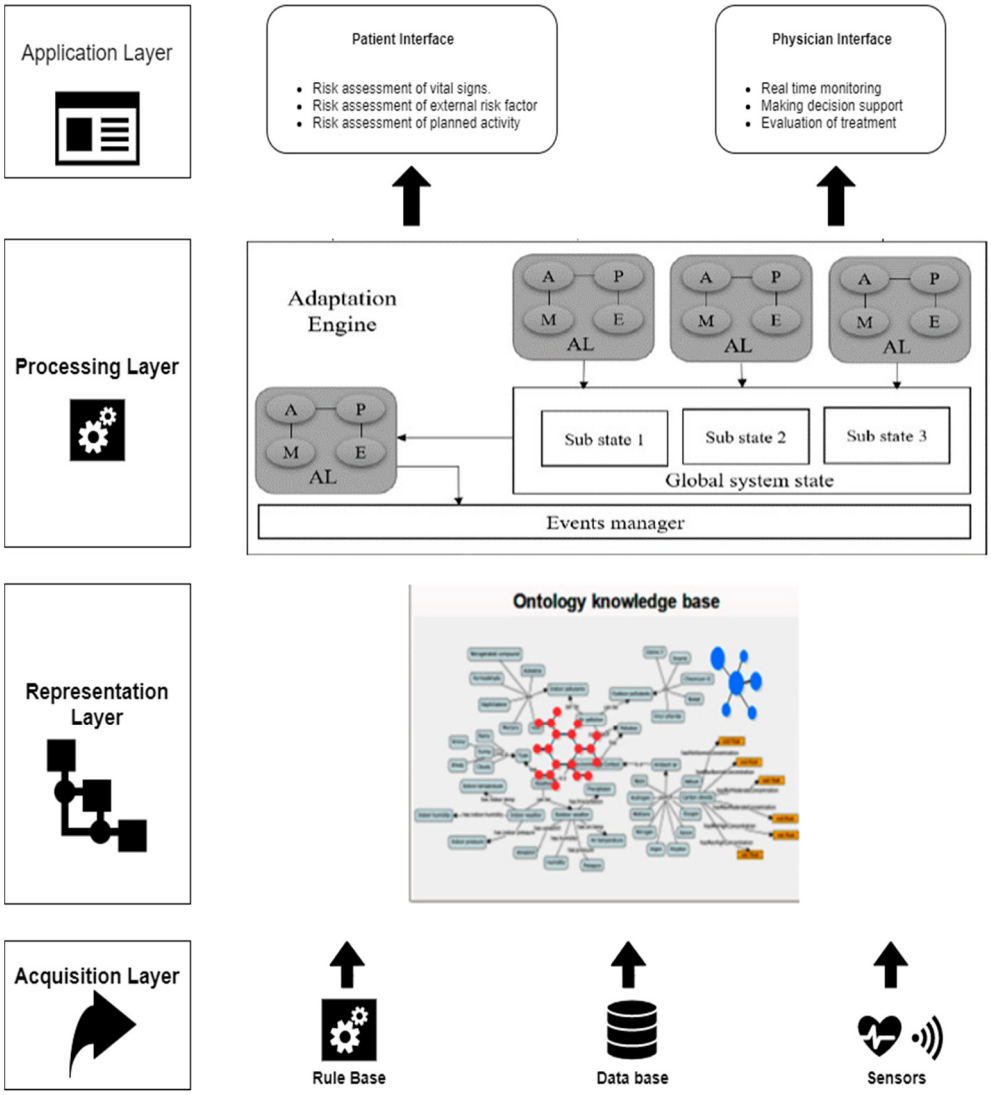


Figure 4. Architecture of COPD context-aware system.

Table 2. Unit categories.

	Biometrics	Environmental	Activities	Location	Duration
Data	Biometrics Data	Environmental Data	Activities Data	Location Data	Duration Data
Rules	Biometrics Rules	Environmental Rules	Activities Rules	Location Rules	Duration Rules
Services	Biometrics Services	Environmental Services	Activities Services	Location Services	Duration Services

The fourth monitoring unit is the location unit, which is responsible for detecting patient location (e.g., indoor or outdoor; home or work; city, mountain or coastal area) and updating its own state with the current location.

The fifth monitoring unit is the duration unit, which is responsible for monitoring the allowed duration for each biometric and environmental factor and storing these data in the duration state.

The central adaptation unit is the core of our system. It is responsible for monitoring the global state, which eliminates the burden of dealing with the continuous streaming of the patient's biometrics and environmental data. By collecting all the contextual data in the

global state and each category in its own substate, we can access all the current biometric and external factor values with their corresponding safe ranges, as well the current physical activity and the planned list of activities.

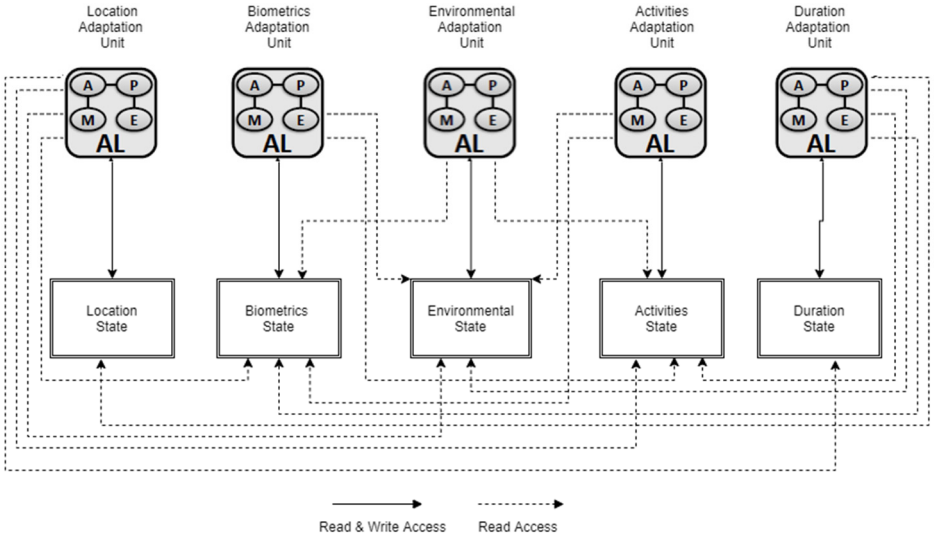


Figure 5. Subunits and substates.

Depending on the previous data, the central unit can detect any potential risk or abnormal situation by comparing the current value of each factor in the substates with its normal range, which has been adapted by every sub-adaptation unit. When an abnormal situation is detected, the central unit detects what action should be taken to prevent an exacerbation in the patient's health state.

The reasoning process (Figure 6) is implemented in this unit following the guidelines set by Ajami and Mcheick [11].

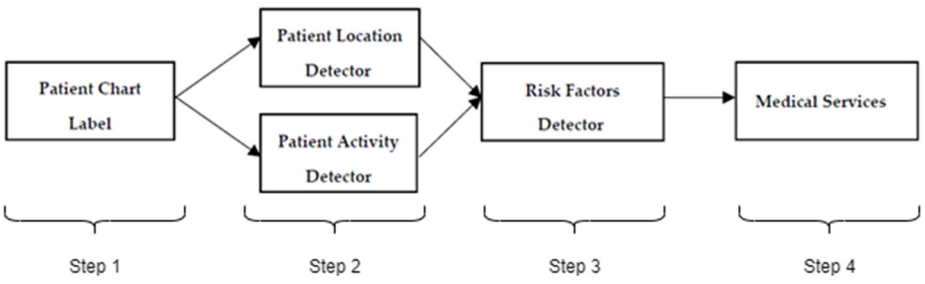


Figure 6. Reasoning process [11].

The patient label chart was explained in detail by Ajami and Mcheick in [11]. It represents an evaluation of the patient's profile and is used to estimate the patient's susceptibility to various external irritants of COPD by classifying the severity of COPD as low, moderate, severe or highly severe based on the rules and measurements of some parameters (factors). These data are retrieved from the patient profile stored in the database.

After retrieving the patient label chart, the patient location and the patient activity that were detected in step 2 are handled by the environment adaptation and activities adaptation subunits, respectively. Next, the process of risk detection is performed in step 3 by comparing the captured environmental and biological factors along with their corresponding safe ranges. Any violation of the adapted safe range triggers the event manager, which in turn detects which service should be activated. The event manager depends on the available data and the rule base, which contains a set of services-related rules [11], to decide which services should be activated. Once the needed services are

identified, the event manager triggers and calls the related components and functions that are responsible for performing those services.

5. Validation of Adaptation Model's Performance

5.1. Implementation and Tools

To test and validate our proposed system, we implemented a simulation web app using a set of JavaScript frameworks and tools. This served as a practical demonstration of how to convert our proposed self-adaptive system's components into an integrated application.

Since our self-adaptive app needed to contain two interfaces (the patient interface and the physician interface), both of which are managed by one core, we implemented our system as a web app. The core system was developed as a Node.js server with a Node.js library that we used to compose our rules engine. For the interfaces, we used React and React Native with Redux as the system state container.

5.1.1. Rules Engine Implementation

We expressed every rule from the chart in JSON (JavaScript Object Notation) standard and data interchange format. Then, this rule is passed to the engine, as shown in the following example.

```
import { Engine } from 'json-rules-engine'
let engine = new Engine()
let rule1 = {
     conditions: {
          all: [{
               fact: 'outdoor-temperature',
               operator: 'equal',
               value: 'severe',
               path: '.profile-severity'
          }]
     },
     event: {
          type: 'update-outdoor-temperature-save-range',
          params: {
               lowest: 18,
               highest: 28
          }
     }
}

let rule2 = {
     conditions: {
          all: [{
               fact: 'outdoor-temperature',
               operator: 'greaterThan',
               value: 65,
               path: '.age'
          }]
     },
     event: {
          type: 'update-outdoor-temperature-save-range',
          params: {
               lowest: 14,
               highest: 28
          }
```

```
    }
}

let rule3 = {
    conditions: {
        all: [{
            fact: 'outdoor-temperature',
            operator: 'between',
            value: [44,64],
            path: '.age'
        }]
    },
    event: {
        type: 'check-vital-signs',
    }
}

let rule4 = {
    conditions: {
        all: [{
            fact: 'outdoor-temperature',
            operator: 'between',
            value: [44,64],
            path: '.age'
        },
        {
            fact: 'outdoor-temperature',
            operator: 'lessThan',
            value: 18.5,
            path: '.BMI'
        }
    ]
    },
    event: {
        type: 'update-outdoor-temperature-safe-range',
        params: {
            lowest: 0,
            highest: 28

        }
    }
}

engine.addRule(rule1)
engine.addRule(rule2)
engine.addRule(rule3)
engine.addRule(rule4)
```

5.1.2. State Units Implementation

After we created the rules engine, we used data obtained from medical records to simulate the streamed data and the patient profile data.

The patient profile data are fetched from the server as soon as the application is launched and are stored in the global state (using Redux), whereas the contextual data are continuously updated in the global state whenever they are received from the sensors.

The following example clarifies how the data are stored in the global state.

```
const defaultAppState: App.State = {
    patientProfileData: {
        gender: 'female',
        age: 56,
        height: 163,
        weight: 71,
        smoker: true,
        goldStage: 'stage2'
    },

    biomarkersState: {
        baselineBiomarkers: {
        heartRate: 69,
        heartrateMax: 152.4,
        SPO2: 97.857,
        PaO2: 73,
        PaCO2: 35,
        diastolicBloodPressure: 84.281,
        systolicBloodPressure: 140.614,
        respirationRate: 31.044,
        FEV1: 1.26,
        VO2: 1.384
    },

    biomarkersDuringLightExercise: {
        heartRate: 108,
        SPO2: 96.857,
        PaO2: 75,
        PaCO2: 42,
        diastolicBloodPressure: 86,
        systolicBloodPressure: 148,
        respirationRate: 36.044,
        VO2: 2.382
    },

    biomarkersDuringModerateExercise: {
        heartRate: 127,
        SPO2: 94.857,
        PaO2: 84,
        PaCO2: 37,
        diastolicBloodPressure: 93,
        systolicBloodPressure: 160,
        respirationRate: 49,
        VO2: 4.114
    },

    biomarkersDuringVigorousExercise: {
        heartRate: 133,
        SPO2: 93.857,
        PaO2: 93,
        PaCO2: 33,
        diastolicBloodPressure: 97,
        systolicBloodPressure: 175,
        respirationRate: 60,
        VO2: 7.007
```

```
        },

        environmentalState: {
            voc: 0.062,
            co2: 702,
            temperature: 20.83,
            humidity: 72.09,
            pm10: 10.2,
            pm25: 9
        }
}
```

5.2. Case Scenarios for Understanding the Separation of Units and the Efficient Execution of the Subset of Rules Based on Changed Attributes

We used a set of COPD rules generated in previous work to create testing scenarios for our system to ensure that the system could apply the rules and adapt the acceptable ranges for each biomarker and environmental factor.

Figure 7 shows the case of a COPD patient who is 68 years old, in the second stage of COPD, has a body mass index of 22.8, has diabetes, has a dyspnea scale score of three and is on two drugs.

Case 1: According to this profile, the patient's PaO_2 level range is 88 to 100 mmHg. However, PaO_2 level is not fixed, as it is influenced by location, weather and activity.

Case 2: Consider a scenario where the patient is at home at 0 m above sea level and begins performing light activity. This state update will trigger the activity unit to send the updated data to the rules engine, which will respond with the recommended action. In this case, the action will update the acceptable range for PaO_2 to 80–100 mmHg. During this process, the biometrics unit will monitor the current value of PaO_2. If there is any break from the safe range limits, an alarm will be triggered.

Case 3: The patient later goes to play sports. This update in the activity state will trigger the activities unit again and run the rules engine, which will respond with a new range of 70–100 mmHg.

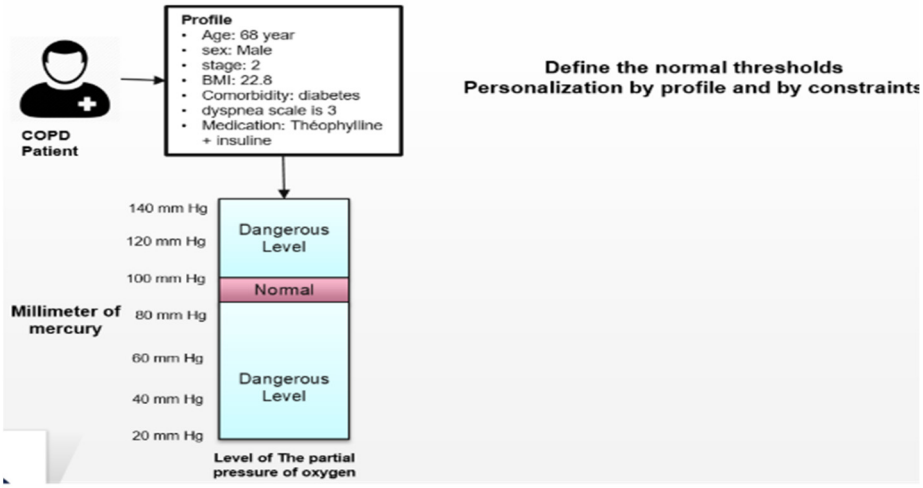
Case 4: The patient reaches a mountainous area with a height of 800 m. This triggers the location unit to send the updated data to the rules engine. These rules respond with a new safe range for PaO_2 (60–100 mmHg). However, this time, the rules engine sends a condition that should be monitored by the time unit. Therefore, case 5: if the patient remains at this altitude while performing heavy physical effort for more than 20 min, as in Figure 7, the patient will be in a dangerous situation. In this event, the safe range for PaO2 will again fall to 88–100 mmHg, and the patient will be alerted if this limit is exceeded.

As can be observed from the above, the system can adapt the acceptable range for PaO_2 levels each time a change occurs in the patient context.

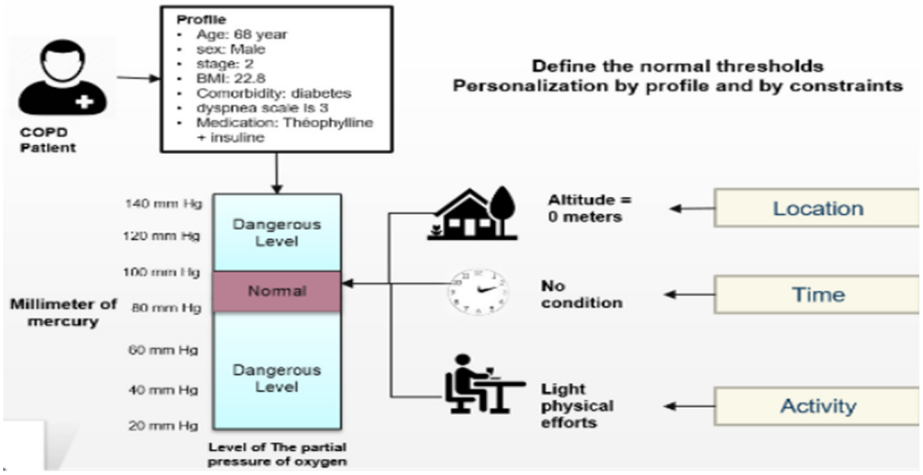
5.3. Results and Discussion

The main focus of the validation process is the efficiency of the proposed system in providing continuous monitoring of patient status and its ability to adaptively apply the required changes to prevent any dangerous exacerbation.

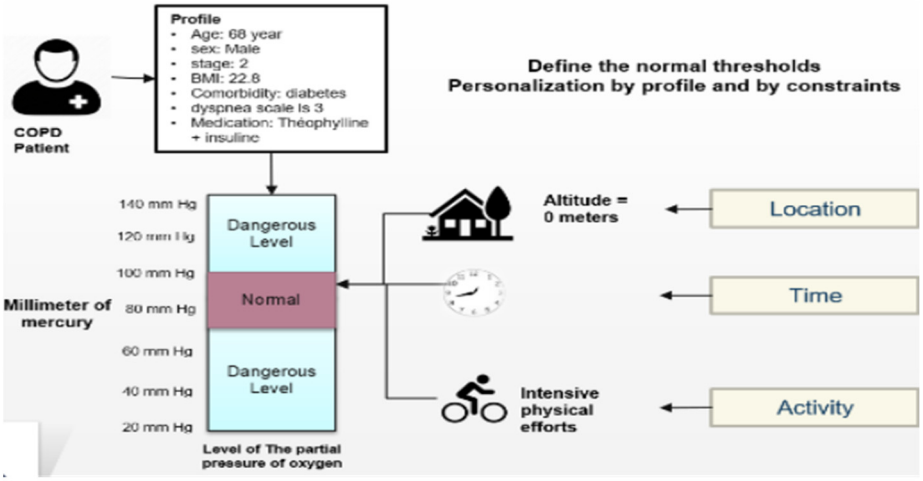
The testing scenarios we performed proved our system's ability to handle the complexity of monitoring the enormous amount of contextual data and keep track of the latest updates in the global state. Indeed, this system is decomposed into 5 subunits and the execution time is reduced from 23,545 milliseconds (hall rules) to 245 milliseconds when we consider 100 rules. In addition, using the separation of concerns (aspect-oriented) approach to design the system facilitated the implementation of the adaptation logic by separating the categories of data that each adaptation unit was responsible for observing.



Case 1



Case 2



Case 3

Figure 7. *Cont*.

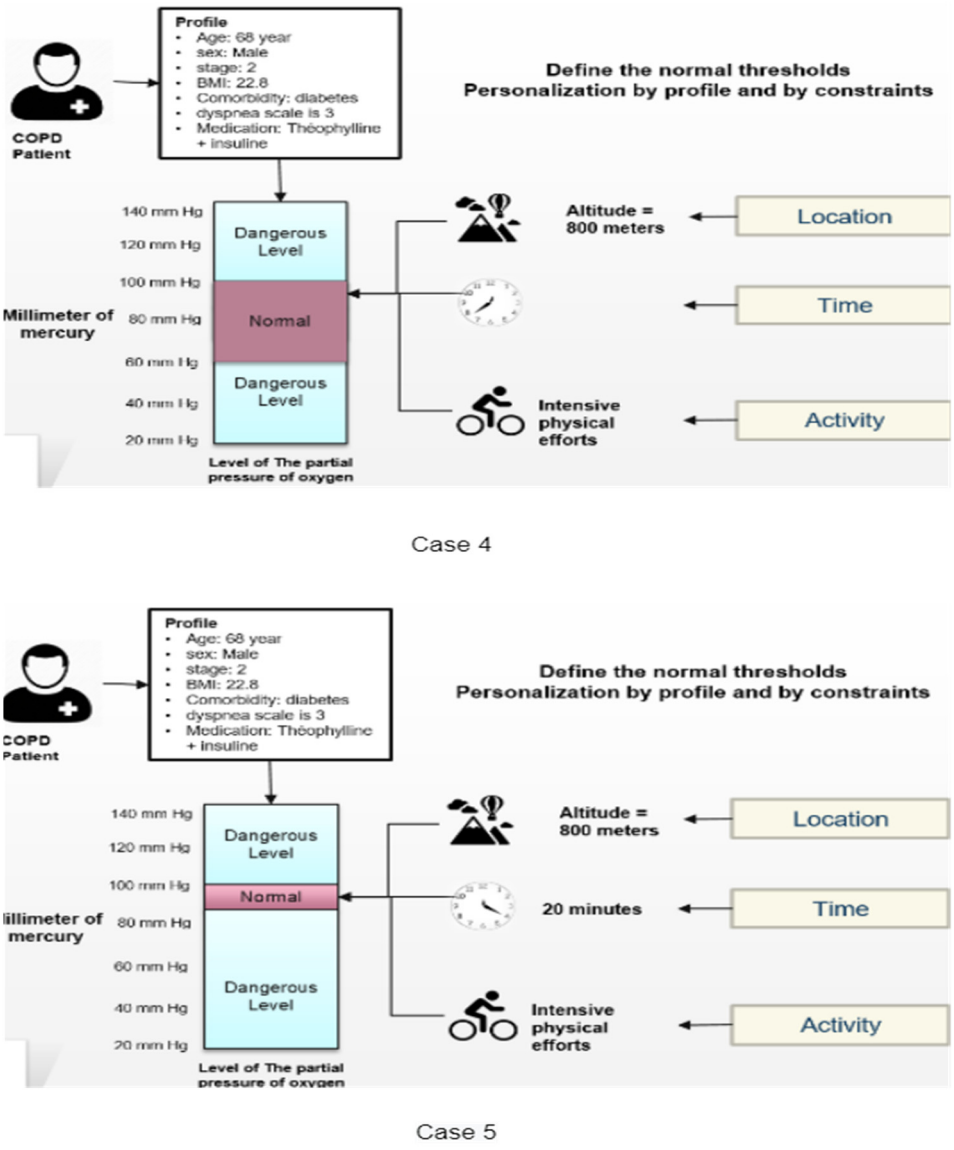


Figure 7. Testing and adaptation scenario.

After testing some rules that led to calling a sequential set of actions and multiple updates in the state units, the system was able to adapt the safe ranges for different environmental and biometric factors and identify the appropriate action to take in an abnormal situation.

Following a decentralized approach by dividing the rules and data into categories reduced the execution time for the rules engine. In this way, rather than running the entire engine with all rules each time there is a change in the context, only the rules related to the change are triggered.

As shown in the following Figures 8 and 9, the execution time for the entire rules engine is 23.545 s, while the execution time in the scenario described above is 0.245 s.

Nevertheless, our proposed system can be tested with more scenarios using the entire rules engine comprised of all COPD rules. Note that we simulated a limited set of scenarios with a limited set of rules (100 rules).

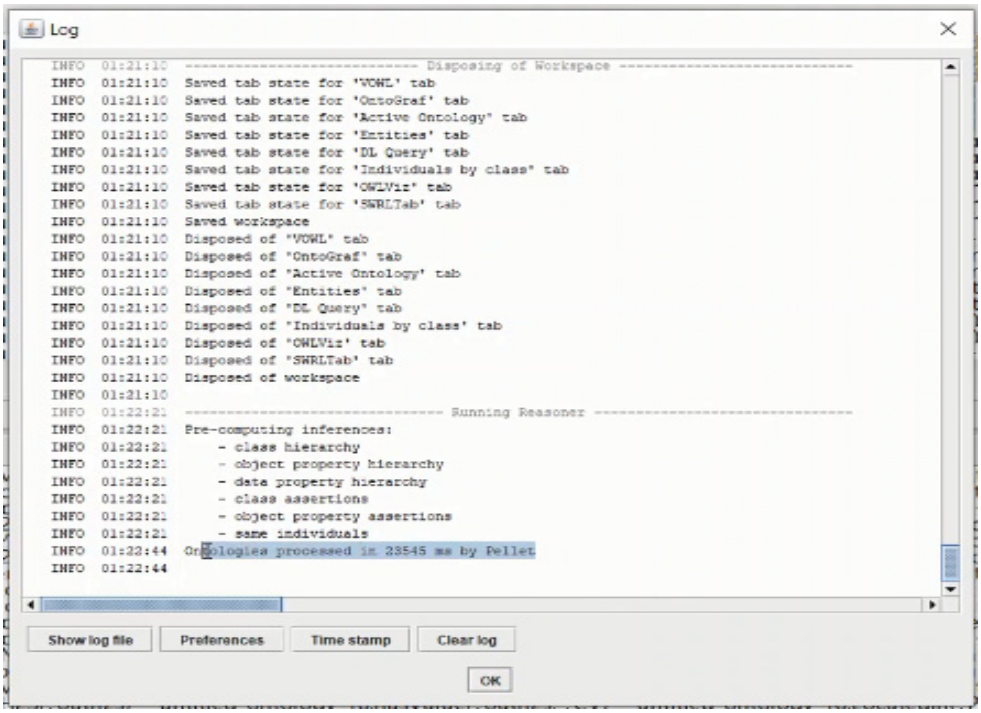


Figure 8. Execution time for entire rules engine, ontologies processed in 23,545 ms.

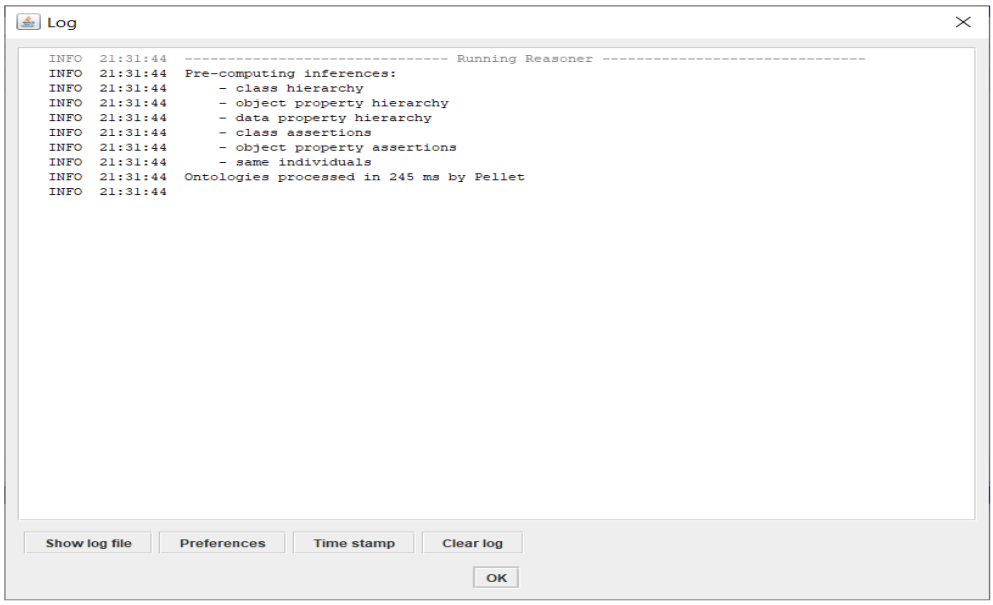


Figure 9. Execution time for the scenario described in Section 5.2, using different units. It is not necessary to execute the entire rules engine—only the rules for one context. It shows that ontologies processed in 245 ms.

6. Conclusions and Future Work

In this paper, we presented a model architecture for a context-aware, self-adaptive system, which we used to design and validate a COPD healthcare telemonitoring system. The system is backed by a medical rules engine in the COPD domain developed in a previous study [12,13], which we used as the knowledge base to determine the safe ranges for patients' biomarkers and external factors and then decide which actions to take to prevent severe exacerbations in patients' health state. Our system was designed and

validated after a thorough analysis of the requirements and based on a taxonomy of the properties of self-adaptive software.

Our main contribution in this work is the development of a context-aware, self-adaptive system architecture that can deal with enormously varied and complex contextual data and different sets of services using decentralized adaptation units. Each of these units is concerned with monitoring a specific kind of streamed data and updating the system state accordingly, and each unit can read the contextual data stored in the state by the other units. This makes monitoring and adaptation easier and less complex by applying the separation of concerns principle. It also improves efficiency and reduces the execution time of the rules engine by applying the set of rules related to the context instead of running all the rules each time the context changes. Future work will integrate complete COPD rules and test the system with real-time data streaming to improve the adherence of the COPD system.

Author Contributions: H.M. proposed and designed the separation of concerns model to improve the performance of our healthcare system. H.M. and J.S. analyzed the architecture and computational model. J.S. wrote the manuscript in consultation with H.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC) and by University of Quebec at Chiocutimi, Chiocutimi (Quebec), Canada.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

References

1. World Health Organization. The Top 10 Causes of Death. Available online: http://www.who.int/mediacentre/factsheets/fs310/en/ (accessed on 3 June 2017).
2. Chronic Respiratory Diseases. Available online: www.who.int/respiratory (accessed on 15 July 2018).
3. Casas, A.; Troosters, T.; Garcia-Aymerich, J.; Roca, J.; Hernández, C.; Alonso, A.; del Pozo, F.; de Toledo, P.; Antó, J.M.; Rodríguez-Roisín, R.; et al. Integrated care prevents hospitalisations for exacerbations in COPD patients. *Eur. Respir. J.* **2006**, *28*, 123–130. [CrossRef] [PubMed]
4. Global Strategy for the Diagnosis Management and Prevention of COPD. Global Initiative for Chronic Obstructive Lung Disease (GOLD). 2011. Available online: http://www.goldcopd.org/ (accessed on 13 January 2021).
5. McKinstry, B. The use of remote monitoring technologies in managing chronic obstructive pulmonary disease. *QJM* **2013**. [CrossRef] [PubMed]
6. Bolton, C.E.; Waters, C.S.; Peirce, S.; Elwyn, G. Insufficient evidence of benefit: A systematic review of home telemonitoring for COPD. *J. Eval. Clin. Pract.* **2010**, *17*, 1216–1222. [CrossRef]
7. Polisena, J.; Tran, K.; Cimon, K.; Hutton, B.; McGill, S.; Palmer, K.; Scott, R.E. Home telehealth for chronic obstructive pulmonary disease: A systematic review and meta-analysis. *J. Telemed. Telecare* **2010**, *16*, 120–127. [CrossRef]
8. Jaana, M.; Pare, G.; Sicotte, C. Home telemonitoring forrespiratory conditions: A systematic review. *Am. J. Manag. Care* **2009**, *15*, 313–320. [PubMed]
9. Bartoli, L.; Zanaboni, P.; Masella, C.; Ursini, N. Systematicreview of telemedicine services for patients affected by Chronic Obstructive Pulmonary Disease (COPD). *Telemed. J. E-Health* **2009**, *15*, 877–883. [CrossRef] [PubMed]
10. Mcheick, H.; Saleh, L.; Ajami, H.; Mili, H. Context Relevant Prediction Model for COPD Domain Using Bayesian Belief Network. *Sensors* **2017**, *17*, 1486. [CrossRef] [PubMed]
11. Ajami, H.; Mcheick, H. Ontology-Based Model to Support Ubiquitous Healthcare Systems for COPD Patients. *Electronics* **2018**, *7*, 371. [CrossRef]
12. Ajami, H.; Mcheick, H.; Mustapha, K. A Pervasive Healthcare System for COPD Patients. *Diagnostics* **2019**, *9*, 135. [CrossRef] [PubMed]
13. Ajami, H.; Mcheick, H.; Mustapha, K. Ubiquitous Healthcare Systems and Medical Rules in COPD Domain. In *How AI Impacts Urban Living and Public Health ICOST 2019. Lecture Notes in Computer Science*; Pagán, J., Mokhtari, M., Aloulou, H., Abdulrazak, B., Cabrera, M., Eds.; Springer: Cham, Switzerland, 2019; Volume 11862.
14. Krupitzer, C.; Roth, F.M.; VanSyckel, S.; Schiele, G.; Becker, C. A survey on engineering approaches for self-adaptive systems. *Pervasive Mob. Comput. J.* **2015**, *17 Pt B*, 184–206. [CrossRef]
15. Oreizy, P.; Gorlick, M.M.; Taylor, R.N.; Heimhigner, D.; Johnson, G.; Medvidovic, N.; Quilici, A.; Rosenblum, D.S.; Wolf, A.L. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intell. Syst.* **1999**, *14*, 54–62. [CrossRef]

16. Elkhodary, A.; Esfahani, N.; Malek, S. FUSION: A Framework for Engineering Self-tuning Self-adaptive Software Systems. In Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, Santa Fe, NM, USA, 7–11 November 2010; pp. 7–14.
17. Mukhija, A.; Glinz, M. CASA A Contract-based Adaptive Software Architecture Framework. In Proceedings of the 3rd IEEE Workshop on Applications and Services in Wireless Networks (ASWN 2003), Berne, Switzerland, 2–4 July 2003; pp. 275–286.
18. André, F.; Daubert, E.; Gauvrit, G. Towards a Generic Context-Aware Framework for Self-Adaptation of Service-Oriented Architectures. In Proceedings of the Fifth International Conference on Internet and Web Applications and Services, Barcelona, Spain, 9–15 May 2010; pp. 309–314.
19. Elmalaki, S.; Wanner, L.; Srivastava, M. CAreDroid: Adaptation Framework for Android Context-Aware Applications. In Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, Paris, France, 7–11 September 2015.
20. Dragone, M. Building Self-adaptive Software Systems with Component, Services & Agents Technologies: Self-OSGi. In *International Conference on Agents and Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 358, pp. 300–316. [CrossRef]
21. Mukabunani, A. Ontology-Based Clinical Decision Support System Applied on Diabetes. Master's Thesis, University of Agder, Kristiansand, Norway, 2017.
22. Humphries, S.M.; Notary, A.M.; Centeno, J.P.; Strand, M.J.; Crapo, J.D.; Silverman, E.K.; Genetic Epidemiology of COPD (COPDGene) Investigators. DL enables automatic classification of emphysema pattern at CT. *Radiology* **2020**, *294*, 434–444. [CrossRef] [PubMed]
23. Ying, J.; Dutta, J.; Guo, N.; Hu, C.; Zhou, D.; Sitek, A.; Li, Q. Classification of Exacerbation Frequency in the COPD Gene Cohort Using DL with Deep Belief Networks. *IEEE J. Biomed. Health Inform.* **2020**, *24*, 1805–1813. [CrossRef] [PubMed]
24. Gaasbeek, J.R.; Martin, J.N. Getting to Requirements: The W5H Challenge. In Proceedings of the 11th International Symposium of the International Council on Systems Engineering, Tel Aviv-Yafo, Israel, 16–17 March 2001.
25. Handte, M.; Schiele, G.; Matjuntke, V.; Becker, C.; Marrón, P.J. 3PC: System Support for Adaptive Peer-to-Peer Pervasive Computing. *ACM Trans. Auton. Adapt. Syst.* **2012**, *7*, 1–19. [CrossRef]
26. Lieberman, H.; Selker, T. Out of context: Computer systems that adapt to, and learn from, context. *IBM Syst. J.* **2000**, *39*, 617–632. [CrossRef]
27. McKinley, P.K.; Sadjadi, S.M.; Kasten, E.P.; Cheng, B.H. Composing Adaptive Software. *IEEE Comput.* **2004**, *37*, 56–64. [CrossRef]
28. IBM. *An Architectural Blueprint for Autonomic Computing*; IBM: Armonk, NY, USA, 2004.
29. Redux Fundamentals, Part 2: Concepts and Data Flow. Available online: https://redux.js.org/basics/data-flow (accessed on 8 May 2021).