Received: 14 October 2023 | Revised: 26 August 2024 | Accepted: 28 August 2024

DOI: 10.1002/spe.3375

RESEARCH ARTICLE

WILEY

A semantically enabled architecture for interoperable edge-cloud continuum applied to the e-health scenario

Angelo Martella[1,2] | **Antonella Longo**[1,2] | **Marco Zappatore**[1,2] | **Beniamino Di Martino**[3,4,5] | **Antonio Esposito**[3]

[1]Department of Engineering for Innovation, University of Salento, Lecce, Italy

[2]Italian Research Center on High Performance Computing, Big Data and Quantum Computing, Italian Center on Supercomputing (ICSC), Casalecchio di Reno, Italy

[3]Department of Engineering, Università della Campania "Luigi Vanvitelli", Aversa, Italy

[4]Department of Computer Science and Information Engineering, Asia University, Taichung, Taiwan

[5]Department of Computer Science, University of Vienna, Vienna, Austria

Correspondence
Angelo Martella, Department of Engineering for Innovation, University of Salento, Lecce, Italy.
Email: angelo.martella@unisalento.it

Abstract

The progress made in the field of medicine and the consequent increase in the prospect of life have contributed to rise people's interest towards a healthier lifestyle. Fitness activity is becoming a must for those who aspire to live more and better. However, this should be accompanied by additional good practices to safeguard individuals' life from risks that could undermine their health. Most of these risks are linked to personal, surrounding, and contextual conditions that technology can detect and monitor. Recommender systems can adequately support fitness activity by performing data analyzes aimed at identifying possible risk factors for users, starting from their physiological data and those related to the closest context where they are. This article introduces the architecture of a recommender system called *App4Health* in the context related to both mobile crowd sensing and wellness. The *App4Health* architecture consists of a smart application platform, capable of interfacing and managing data from heterogeneous edge sources, such as mobile phones, IoT, and sensors. The analysis result consists of the semantic generation of healthy behavioral conducts to the user as Telegram BOT messages. For evaluating the proposed solution, the article also provides a case study and a testbed. The testbed consists of a comparative stress test of two edge software components of the *App4Health*'s architecture in order to identify the performance degradation threshold of these components, assuming that they can be deployed on edge-level hardware devices with different technical specifications and configurations.

KEYWORDS

edge computing, edge-cloud continuum, healthcare recommender system, mobile computing, mobile crowd sensing, semantic interoperability

Abbreviations: BSN, body sensor network; CS, citizen science; CXT, context; GE, generic enablers; H-IoT, Healthcare Internet of Things; HSL, hybrid storage layer; HSMS, hybrid storage management system; IoT, Internet of Things; JSON, JavaScript Object Notation; MCS, mobile crowd sensing; MCSO, mobile crowd sensing ontology; MSA, micro-service architecture; NGSI, next generation service interface; REST, representational state transfer; RS, recommender system; SBC, single board computer; SSN, semantic sensor network; SSNO, semantic sensor network ontology; SSWAD, semantic sensors Web API description; WSN, wireless sensor network.

1 | INTRODUCTION

The evolution of recommender systems (RSs) is undergoing a new impulse, especially in terms of potential areas of application.[1] Originally, RSs aimed to provide users with personalized online product or service recommendations to handle the increasing online information overload problem and improve customer relationship.[2] These systems utilized personal, implicit, and local information to provide tailored suggestions.[3]

RSs have been applied in various fields, including movies, songs, online courses, restaurants, tourist places, news, websites, and blogs, but mostly they continue to be developed nowadays .[4,5] RSs often utilize multiple algorithms to enhance user experience and system accuracy.[6] Recent developments in Internet of Things (IoT) and mobile phone technologies have significantly contributed to extend the applicability of RSs to a wide range of domains by providing value-added services to users, through more general-purpose recommendations.

The study of innovative and high-quality RSs is gaining more and more interest and new cross-domain approaches are emerging to design RSs capable of generating recommendations obtained by relating data from different application fields. Significant contributions offered by the internet and smart devices are opening new frontiers in terms of data availability at disposal of RSs. Recently, a new category of RSs has emerged with the name of IoT-based RS. These RSs consume data produced by IoT devices to generate recommendations for their users. Data gathered and communicated by IoT devices may include data specific to the system domains, along with additional data coming from complementary domains that can be used to refine the process of generating recommendations.

However, these significant achievements have potentially contrasting implications. On the one hand, IoT devices, but mostly smartphones and wearable devices, represent innovative and additional data sources, capable of offering a wide and more detailed information framework.[7,8] Specifically, wearable sensors can make available relevant data about the user's state of health and exercise.[9–11] So, data directly provided by users can be enriched by additional contextual data that are automatically detected and retrieved by sensors. The resulting data include the vital parameters of a user, but also those linked to the corresponding environmental conditions surrounding the same user. Contributions offered by the internet and smart devices are able to implement a dynamic network of objects/resources, which unfortunately is also loosely-coupled, and decentralized.[12,13] Anyway, considering the potential capabilities that each object/resource can offer in terms of sensing, computing, communication, and actuating, the resulting network can constitute an instance of ubiquitous computing.[14] As a consequence, RSs are proposing themselves as candidates to ubiquitously support most of the activities that characterize and define our everyday life. They aim at collecting data related to the daily activities a user decides to be monitored, in order to have suggestions and insights on possible best practices.

On the other hand, except for the crucial dangers to the privacy of individuals,[15,16] RSs have at their disposal amounts of data that are markedly fragmented and heterogeneous in terms of formal and substantive representation.[17] These data characteristics have a significant impact on the development possibilities of RSs, which aim to support cross- and inter-domains capabilities.[18–20] Studies in References 21 and 22 emphasize the importance of data quality, particularly rating data characteristics and item content data completeness. In particular, in Reference 21 authors further emphasize the influence of data characteristics on recommendation algorithm performance. While, in Reference 23, data mining techniques are used for generating recommendations, underscoring the critical role of data characteristics in RSs. Another aspect which is crucial for cross-domain RSs is given by shared standards. They are essential because they can facilitate knowledge transfer between different domains, improving their performance.[24,25] Consistent information transfer is crucial for ensuring the accuracy of recommendations in the target domain.[26] Cross-domain collaborative RSs can suggest items related to multiple domains, underscoring the importance of shared standards in this context. Overall, shared standards play a crucial role in enhancing RSs.[27] The growing amount of data, along with the number of corresponding data services, are constantly and continuously complicating the range of challenges an RS has to address. Also for these reasons, reference standard specifications on how to design the architecture of an RS is yet to come, although the research activities in this context are particularly fervent.[28] A key enabler towards the development of cross- and inter-domain RSs is represented by the definition of a shared standard for describing sensor interfaces. By proceeding in this way, it becomes possible to implement a seamless interoperability among devices and to reduce the need for adopting specific adapters.[29,30]

To address interoperability issues, a possible approach is to apply semantic technologies for describing Web APIs, including APIs related to sensors.[31] According to this, it becomes possible to discover and compose the corresponding exposed functionalities in order to implement even complex applications. In addition, the informative framework for an application can be extended by integrating various domain ontologies corresponding to the monitoring sensors used within the different contexts.

In summary, beyond adequately facing data privacy crucial aspects, the design and implementation of an IoT-based RS must address issues related to the following: (1) the formulation of recommendations should adopt the use of specific data management techniques to effectively tackle big data features like speed, heterogeneity, and quantity, (2) the data availability for an RS has to be expanded by providing a wide range informational framework to be used for a more dynamic and adaptive formulation of recommendations, and (3) the edge-cloud continuum (ECC) emphasizes the need for distributed IoT data processing, necessitating the implementation of specific data security policies for the RS to ensure reasonable and acceptable system service levels.

Starting from such a scenario, this article proposes a semantically-enabled architecture of an inter-domain RS called *App4Health*, which is based on both IoT technologies and mobile crowd sensing (MCS). For implementing interoperability, *App4Health* adopts the ECC approach applied to the e-health scenario. Indeed, the reference domain for *App4Health* is obtained by integrating monitoring data related to both personal wellness and contextual conditions. The current version of the *App4Health* platform aims at addressing challenges related to big data management and device heterogeneity. Indeed, *App4Health* platform represents the evolution of a previous architecture proposed in Reference 32. Other key aspects corresponding to trust management, privacy, security, and quality of service are also planned to be faced in the future evolution of *App4Health* RS.

The architecture of the cross- and inter-domain RS we propose allows to investigate the following Research Questions (RQs):

RQ-1 How to use IoT and semantic technologies to design a possible architecture of an RS that uses IoT and semantic technologies in order to formulate best practices to suggest to its user? Section 3 gives an answer to this RQ.
RQ-2 How to collect, process, and analyze data from MCS scenarios as well as personal sensing contexts to make it possible for users to monitor and preserve their health? This RQ is addressed in Sections 4.1.1 and 4.2.1.
RQ-3 How to support the formulation of best practices for users by reducing both the internet and cloud dependencies in order to provide recommendations/suggestions according to contextual conditions and in near-/real-time? This RQ is tackled in Section 3.

The present work is structured as follows: Section 2 outlines the research background concerning the MCS paradigm, and the ECC approach, as well as the current state of the art in terms of RSs. Sections 3 and 4 respectively report the overall architecture and a detailed discussion of each tier and layer the proposed *App4Health* platform provides. Section 4 also provides a specific discussion about the reference ontologies the *App4Health* platform uses to formulate suggestions and/or recommendations to propose to its users. Section 5 outlines and discusses the *App4Health* platform evaluation, by proposing an edge testbed using two different hardware and software configurations. This section also includes a case study and the answers to the three RQs we have previously introduced. After analyzing the evaluation results, some final considerations and lessons learned are formulated in Sections 5.4 and 5.5, respectively. Finally, Section 6 reports conclusions and future work that we set out to achieve.

2 | BACKGROUND AND RELATED WORKS

2.1 | Mobile crowd sensing

The MCS paradigm consists of communities of users that share common interests, interface the same data sources, and use the sensors that their mobile devices make available, directly and indirectly, in order to gather large-scale observations of heterogeneous phenomena.[33] Initially, the first approach to MCS was called crowdsourcing and adopted web technologies for implementing software platforms (e.g., Amazon Mechanical Turks[34]). These platforms consider each crowd worker as a mobile computing unit, completing tasks using his/her mobile devices[35] and relying on spatial information like location and mobility. Mobile crowdsourcing applications span various aspects of everyday life, including services for real-time taxi-calling like Uber,[36] supermarket product placement checking like Gigwalk[37] and TaskRabbit,[38] on-wheel meal-ordering like GrubHub[39] and Instacart,[40] and citizen sensing like Waze[41] and OpenStreetMap.[42]

The advent of mobile internet and sharing economy have led to a shift from the web-based to mobile (or spatial) crowdsourcing platforms[43] giving rise to the phenomenon of MCS. The success of the MCS paradigm is largely due to the widespread use of smartphones and wearables, which have a rich array of built-in sensors, including accelerometers,

gyroscopes, GPS, microphones, and cameras. These sensors are commonly used in various applications such as healthcare, environmental monitoring, and traffic monitoring.[44] Examples of MCS applications for promoting healthy eating are HealthAware,[45] MPCS,[46] and DietSense.[47] Nericell[48] is an MCS initiative for monitoring traffic conditions. GasMobile,[49] HazeWatch,[50] and ThirdEye[51] are MCS projects for air pollution monitoring, while Garbage Watch[52] and WasteApp[53] aim to improve the recycling process by monitoring the content of recycling bins.

The potential pervasiveness of MCS can be realized on the direct initiative of device owners to participate in data collection (i.e., *participatory MCS*) or by delegating it to a dedicated mobile application (i.e., *opportunistic MCS*). MCS has the potential to revolutionize data collection, but it faces challenges in terms of data reliability, participant incentivization,[54] crowd monitoring using mobile phones,[55] advanced time/space localization techniques,[56] and software architecture for smart city environments.[57] These studies emphasize the need for further research and development to address these limitations and maximize MCS' potential, highlighting the need for improved localization techniques and back-end systems for urban and indoor scenarios. MCS in consumer apps and devices presents both opportunities and challenges. In Reference 58, a service-oriented approach is proposed to improve user experience, while in Reference 59 a game theory approach is discussed to enhance consumer experience by addressing issues like cost sensing and data quality. In References 60 and 61, usability and user-friendly design and user privacy and data trustworthiness are highlighted as crucial for MCS applications, respectively.

By integrating with IoT platforms,[62] MCS can be a powerful enabler of Citizen Science (CS) activities[63,64] and can represent an important asset in urban sensing in terms of environmental monitoring,[65] infrastructure control,[66] traffic congestion assessment,[67] and management of public facilities.[68] Data collected via MCS campaigns constitute a precious source to exploit for deriving data-driven insights on pollution levels, which are particularly useful for evaluating possible actions of pollution mitigation or abatement. Unfortunately, users are relegated to having little room for action with respect to the effective improvement of their quality of life, considering that the contribution they offer must aim at something more than mere data collection. Moreover, just a few MCS solutions currently present in scientific literature are capable of both combining multiple sensing functionalities and providing recommendations to users on how to behave or how to react to specific situations that are inferred through the observation and/or the processing of the collected data.

The general design approach adopted by these solutions is to implement MicroService Architectures (MSAs), essentially thanks to their capability to enable the connection between MCS and CS, in a fully integrated way. Such an approach makes it possible to exploit data models and scalable architectural solutions aimed at the coexistence of distributed computing, edge-localized processing and cloud computing virtualization. According to the *osmotic* approach,[69] MSAs are used to decentralize and distribute data storage and computation from the cloud towards edge nodes. In this way, *osmotic computing* creates the conditions for a multimodal crowdsensing,[32] in which various and numerous IoT devices and sensing-capable devices exist at the network edge. Data collected from the environmental monitoring performed by these devices are stored and processed by MSAs, even by referring to complex pipelines. These pipelines must perform data tasks related to their storage, transmission, processing and analysis, in accordance with precise and fundamental requirements concerning quality assurance, scalability, durability and interoperability.[70]

Regarding this last requirement, an even more urgent and critical challenge is emerging due to the number and heterogeneity of IoT devices: different notations and vocabularies for sensors and parameters, variable data exchange policies, and heterogeneous data representation schemes represent crucial factors for the aforementioned requirements. A rising approach for interoperability is semantics.[71] Among the numerous ontologies already present in literature, it is worth mentioning the standard semantic sensor network (SSN) ontology proposed by W3C as a suitable modeling solution that can be easily extended thanks to its light-weighted modular structure (see Section 4.2.1). Unfortunately, the same cannot be said for possible ontologies related to MCS. For this reason, an MCS ontology aimed at filling the existing modeling gap has been implemented and that is discussed in Section 4.2.2.

2.2 | Edge-cloud continuum

The ECC approach is implemented through a series of hardware and software layers close to the edge of the network, which aims to constitute a federation of agents. In accordance with osmotic computing, the trusted federation to be implemented includes hardware and software agents that allow the computational and storage capabilities to be decentralized towards the edge of the internet, with respect to a traditional cloud-based system. While hardware agents are represented by special edge devices, software agents correspond to microservices available near the places where IoT

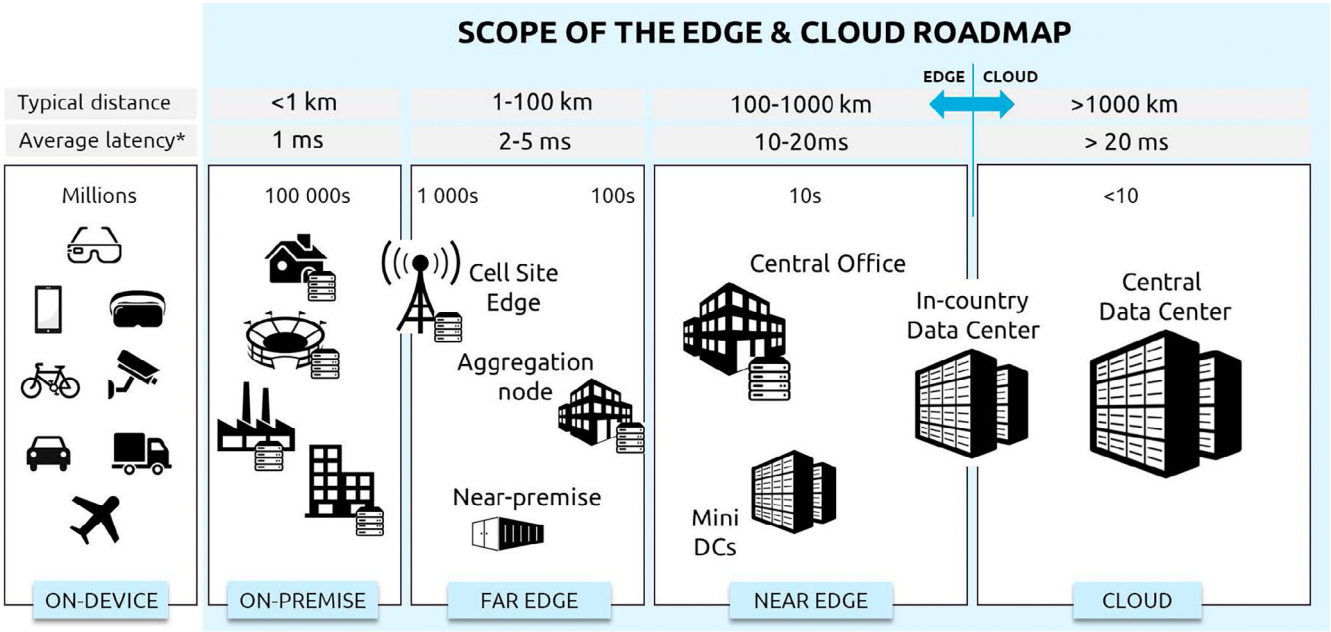

FIGURE 1 Edge-cloud continuum: European industrial roadmap.[72]

data is produced. Therefore, these data could represent digital transpositions of the objects of reality, which in literature are referred to as digital twins. According to the paradigm of edge computing, the trend is to process data more and more in the proximity of the place where they are generated and collected. The main goal is to extend cloud computing in order to provide a more holistic and suitable platform for innovative services and applications. Edge computing includes any device that is at the edge of the core network. An intermediate solution to cloud and edge computing is fog-computing. This approach involves implementing a computational and storage layer between the cloud and edge computing layers, in an attempt to make information and services available in near real-time. Therefore, edge computing represents only one of the components envisaged by the ECC model. To adequately implement the ECC, it is essential to adopt an architectural model for the software platform that makes it possible to organize computing capabilities and network functions, in dynamic environments characterized by particularly flexible processes. A transversal platform capable of fully supporting storage and computing features from the edge to the cloud tier is a fundamental requirement for any sustainable digitization strategy. Figure 1 shows the European proposal to ECC which can be considered particularly relevant for encompassing a detailed spectrum, ranging from on-premise edge to far edge (within 100 km of customers and premises), near edge (typically hundreds of kilometers away from the customer), and even regional data centers.[72]

2.3 | Recommender systems

The concept of RSs was initially introduced in the internet context to identify systems capable of supporting users in searching relevant information with a significant reduction of time and effort. With the advent of e-commerce, these systems became popular as smart computer-based techniques that analyze users' adoption and usage before proposing them appropriately selected items to buy. In general, item selection performed by RSs takes into consideration a wide and accurate choice task among an unspecified set of possible candidates. The main RS objective was to support users in their decision making while purchasing items, by providing them useful and accurate recommendations.[73] The promising results obtained by e-commerce RSs and the emergence of new and captivating technologies have aroused strong interest among researchers in their reuse also in other application domains, including books,[74] movies,[75] music,[76] e-learning,[77–79] fitness-health, and tourism.[80] According to this original connotation, a RS can recommend items to a user using a combination of approaches, such as collaborative filtering (CF), content-based filtering

(CBF), demographic-filtering (DFF), and knowledge-based (KB). A CF RS involves collaborating opinions of other users to make recommendations, while a CBF RS recommends items based on user profiles. A DFF RS supposes that users with similar preferences tend to have similar tastes, and finally a KB RS considers user needs to recommend items. CF systems can be further classified into memory-based and model-based systems, with memory-based systems using user ratings for items and model-based systems building models based on user preferences and other factors.[81,82] The rapid and disruptive diffusion of this phenomenon has favored a new declination of the RS concept as an innovative and direct communication means between users and services providers. Although the operating principle of an RS is roughly the same regardless of the application domain, the design and development of such a system can adopt even deeply different approaches and technologies. These approaches and technologies are borrowed from research fields related to artificial intelligence, forecasting and approximation techniques, semantics, data analysis, and marketing.[73]

Nowadays, RSs provide specific techniques aimed at providing value-added services to users through more general-purpose recommendations. The study of innovative and high-quality RSs is gaining more and more interest and some new wider approaches within the RSs design are emerging. One of these involves the adoption of a cross-domain approach capable of generating recommendations obtained by relating data from different application fields. Another promising approach provides for the enrichment of domain data with additional data coming from social networks and/or from the user's reference context, in order to derive recommendations that are as dynamic, characterized and contextualized as possible. Despite significant developments, current RSs need further analysis and studies before formulating recommendations that are derived using complex reasoning techniques based on the integration and aggregation of data coming from complementary and multiple domains. Therefore, RSs are proposing themselves as candidates to ubiquitously support most of the activities that characterize and define users' everyday life. They aim at collecting data related to the daily activities a user decides to be monitored, in order to have suggestions on possible best practices. One of the most advanced and innovative research branches within the RS domain plans to adopt a cross-domain approach to the process of generating recommendations,[83] in an attempt to widen the margins of integration and interoperability of current RSs. Below is a review of the main approaches to the RSs development that can be considered as relevant for the solution proposed in this article.

2.3.1 | Knowledge-based recommender systems

Knowledge-based RSs involve formulating recommendations for users by referring to interoperability, integration and querying techniques of knowledge models relating to the reference domain data for the system itself.[84] By referring to integration and interoperability techniques on the semantic models, an RS can implement the inductive and deductive process of the knowledge base through which the recommendations to provide to the user are formulated, in strict compliance with the user's preferences.[85] In literature, an alternative approach to rigorous compliance with user's likely preferences is still possible: recommendations can be formulated by applying some constraints on the same user's likely preferences.[86] In this case, a set of alternatives that are closely to the preferred recommendation is proposed to users, but exclusively when no closely matching recommendations are formulated by an RS system. When knowledge-based RSs use ontologies for knowledge representation (i.e., *ontology-based RS*), proper approaches for defining ontological models related to the domain entities for the RS are needed, usually leveraging the Semantic Web technologies.[87,88] Nevertheless, although a knowledge-based RS is capable of representing and formalizing its reference domain data (which could become very complicated using conventional and traditional approaches), the corresponding knowledge management and modeling techniques can, in turn, become very expensive. In literature, various implementations of knowledge-based RSs are present. García-Crespo et al.[80] propose a semantic RS for hotels that is based on a fuzzy logic applied to data related to their consumers' experiences. Data corresponding to both hotel and customer entities are taken into consideration for proposing recommendations. Dong et al.[89] provide an architecture for a service-concept RS that adopts a semantic similarity model obtained as integration of techniques that adopt two metrics: an ontology structure-oriented metric and a concept content-oriented metric. Mohanraj et al.[90] introduced the ontology-driven bee's foraging approach that attempts to properly predict the web pages that most likely a user will consult. The self-adaptive system aims at tracing the changing needs of online users by referring to an ontological framework, based on similarity comparison and scoring algorithm.

2.3.2 | Context-aware recommender systems

In literature, various definitions of context exist. One of these defines the context as "the situation in which an event happens or the conditions that exist when the event happens" to underline that conditions and situations can change the perspective of an event.[91] Another available definition of context cites "the identity of items or users, for example, their locations, time, activities or any changes that can happen to them."

Starting with these definitions, it is possible to introduce the concept of Context-Aware RS as a multidimensional RS that formulate recommendations by considering its domain data, along with those corresponding to the user context. Today, numerous and heterogeneous sensors exist that can detect the environment surrounding a user, in order to gather and transmit them to an RS.

Context information can be classified into explicit and implicit.[92] While explicit information is directly available when a user decides to share it with the system, such as (birthdate, gender, etc.), implicit information is generally and automatically gathered by sensors, in terms of location, time, temperature, user's preferences, health and so forth. Contextual information can be crucial for RSs that aim at generating more accurate and specific personalized recommendations.[93,94] For this reason, it becomes essential for an RS to take into consideration contextual information during the formulation process of user recommendations.[95] Doing so, it is possible for an RS to be considered as ubiquitous and to generate more dynamic, accurate and adaptive recommendations for users.

2.4 | Fitness and health-related recommender systems

By exploiting IoT technologies, a RS represents an ubiquitous and widespread fitness and health-related solution. This specific RS typology is capable of suggesting recommendations for therapies, dietary regimes, and fitness activities, with the aim of becoming a long-term assistant for its users.

Yong et al.[96] proposed a smart gym assistant that monitors the health status of exercisers. The system collects exercise data by using sensors and fitness bands, before referring to artificial intelligence technologies to formulate recommendations for users' bodybuilding. Nag et al.[97] proposed a double-layer RS that acts as a nutritionist assistant for controlling sodium intake. The reference layers for the solution refer to the exogenous and endogenous contexts. The former is characterized by the contextual data, such as weather stations, pollution monitors, traffic data, and so on. While the latter focuses on the physiological status of the users' body, in terms of data collected by IoT sensors and devices, such as temperature, blood pressure, heart rate, and so on. Casino et al.[98] provided the architectural details of a context-aware RS that collects IoT sensors' data in real-time, before classifying them into health-related information and users' preference information. The RS is integrated with the city sensing infrastructure to provide citizens with route recommendations according to their health conditions and preferences.

Aipe et al.[99] presented a stacked deep learning model for sentiment analysis from the medical forum data. The model uses both convolutional neural networks and long short-term memory to implement a probabilistic model capable of recommending treatments or procedures for a particular disease or health condition. Almeida et al.[100] proposed an RS obtained by combining collaborative and context-based filtering techniques to discover cohorts of interests in order to respectively detect similar users' profiles and generate better suggestions and recommendations. A similar technological approach was followed by Jie et al. to develop a Traditional Chinese Medicine RS.[101] The system's recommendation process is based on users' preferences, content characteristics, contextual data, and demographic data. The recommendation generation refers to the K-nearest neighbors algorithm for identifying users' similarity. Finally, Mojarad et al. developed[102] a context-aware adaptive recommender to provide personal well-being services. The proposed context-aware RS uses the recognized human behaviors to define different contexts, including location, object, frequency, duration, and sequences of frequent activities. An ontology, called Human ActiVity Ontology is then used to conceptualize human activities and their contexts. A set of probabilistic rules is finally used to formulate adaptive recommendations. Table 1 provides a comparison overview in terms of key features between the platform we propose and the other fitness and health-related RSs introduced in this section.

It is now possible to discuss the limitations of the state-of-the-art solutions that we aim to bridge with our proposal. First, all the projects mentioned so far share the same limitation: they just insist on a vertical application area and not to a larger and more intertwined perspective. A further key aspect in favor of the solution we propose regards the choice to possibly opt for data streams as an alternative to data storage. Another significant gap that exists in literature is the shortage of solutions that combine sensing functionalities to the subsequent formulation of user suggestions on how to

TABLE 1 App4Health versus other fitness and health-related recommender systems.

Authors	Goal	Data collected	Recommendation type	Technologies used
Yong et al.[96]	Smart gym assistant	Exercise data	Guidance information for users' body building	Artificial intelligence
Nag et al.[97]	Nutritionist assistant	Contextual data and physiological values	Sodium intake control	Multi-modal user vector
Casino et al.[98]	Health-care assistant	Contextual data and user health conditions and preferences	Route recommendations	Collaborative filtering methods
Aipe et al.[99]	Health-care assistant	Forum and social media posts	Recommending treatments or procedures for a disease or health condition	Deep learning models
Almeida et al.[100]	Cohorts of interest discoverer	Collecting user's preferences	Similar datasets or publications involved in a clinical study	Collaborative filtering, content-based retrieval techniques and semantics
Jie et al.[101]	Traditional Chinese medicine assistant	Users' preferences, content characteristics, contextual data, and demographic data	Personalized and healthy information	K-nearest neighbors algorithm
Mojarad et al.[102]	Personal well-being assistant	Human activity, location, and context objects	Personal well-being services	Machine-learning models and semantics
App4Health	Wellness and health-care assistant	Contextual data and physiological values	Healthy best practices and suggestions	Semantics- and rule-based approaches

perform a specific task or how to intervene in a given situation, depending on collected data. Moreover, the solution we propose is capable of formulating best practises/suggestions using both semantics- and rules-based approaches. In the latter case, best practises/suggestions can be also formulated in near real-time. To the best of our knowledge, none of the research works in the wellness-related context we have mentioned so far provides semantic models for IoT sensing, specifically tailored to infer environment and wellness relationships by interacting and integrating different ontologies.

One additional innovative aspect of the solution we propose is related to the adoption of the Fiware context broker.[103] Fiware[104] is an open-source platform designed and managed by a fervent developers' community. This community has defined a universal set of standards for supporting context management also in terms of smart applications development in various scenarios, including smart cities and smart energy grids. The Fiware core aspect is the context management that is essentially performed by the context broker, which also represents its core component. Around its core component, Fiware provides a set of alternatives as context data/API management block and processing, analysis, and visualization block, along with a powerful and wide interface with IoT third-party systems. By combining the context broker with external components, such as *Cygnus*,[105] it is possible to support data management and analysis, along with the storage of historic data.

Fiware also provides deployment tools. Context information gathered by the context broker can then be made available to other components of the architecture in order to be able to exploit or enrich them. In addition to the context broker, the platform made available by Fiware can be completed with a series of software components called generic enablers (GEs).[106] Beyond being all open and royalty free, these components can almost completely implement a platform for the development of smart applications.

Fiware community participates in various initiatives that can represent valid solutions to common requirements in the context of the application we propose. An example in this sense is the i4Trust project,[107] a new platform for data space management. i4Trust aims to create a sustainable ecosystem in which companies can create innovative services and abandon traditional silos data structures, in favor of sharing, reusing and exchanging data resources. Finally, Fiware context broker supports rule-based notifications as subscription defined on the state of the monitored real-world assets.

App4Health platform adopts this approach to formulate best practices/suggestions for users at the edge of the internet, proposing an alternative to the semantic-based recommendation formulation. Section 5.3 reports a case study in which Fiware context broker is used to develop a rule-based recommendation formulation. A case study on semantic-based recommendation formulation, instead, was already proposed in Reference 108.

3 | ARCHITECTURE IN THE LARGE

The platform to develop represents a RS that uses semantics and rule-based approaches for providing to the users best practices and suggestions to preserve and take care of their wellness and health status. For this aim, *App4Health* monitors the user's vital parameters and environmental conditions of the context in proximity to the place he/she is. For instance, *App4Health* could suggest in advance a user to not move from an indoor place into another which is near where he/she is. The reason for such a suggestion may be due to a poor reconciliation between the values of the user's vital parameters detected by personal smart devices and the environmental conditions of the places near the user's geographical position. After this short *App4Health* introduction, it is possible to proceed with the elicitation of the requirements it must meet.

3.1 | Requirement analysis

Before going into details, a first relevant consideration to do regards the design of the *App4Health* logical architecture having to be obtained as an enhancing evolution of the osmotic computing infrastructure that implements the microservice approach for urban pollution monitoring which was introduced in Reference 32. Osmotic computing represents a paradigm that integrates edge and cloud resources and can constitute an efficient solution for IoT services and applications.[109] It presents challenges in QoS and performance monitoring, which can be addressed through an integrated monitoring system.[110] The paradigm also requires a smart orchestration architecture, such as the MicroELement, to enable the deployment and migration of applications across different layers.[111] One of the most relevant aspects of the previous architecture version was the prerogative of being an open architecture, with the broadest meaning of the term. This prerogative remains valid also for the present version of the architecture. According to the previous architecture, the *App4Health* platform still remains a MSA implemented using Java as a programming language for developing the corresponding microservices. Furthermore, the reference communication formats and architectural style among microservices and devices remain the same: *JavaScript Object Notation* (*JSON*) and *REpresentational State Transfer* (*REST*).

In the following, the previous version of the *App4Health* architecture has to be considered as the osmotic computing infrastructure proposed in Reference 32. Considering this premise, the following requirements and the corresponding solutions the *App4Health* platform must meet are borrowed from its previous version. (a) It must be an open architecture, (b) it must efficiently and widely support device heterogeneity, (c) it must be based on both IoT technologies and MCS, (d) it must implement a MSA, and (e) it must adopt standard communication protocols. Instead, the additional requirements that the revised *App4Health* platform must specifically met are reported below: (1) when possible, it must be developed using the same approaches and technologies already adopted for the previous version, (2) it must adopt a semantically-enabled architecture of an inter-domain RS, (3) it must make available an informative domain which is obtained by integrating monitoring data related to the user's physiological state and the corresponding contextual conditions, (4) it must be compliant with the ECC approach for implementing interoperability, (5) it must redevelop the middleware layer of its previous version by adopting a context broker solution (instead of a message broker one) and by implementing a data lake instance, (6) it must formulate recommendations for registered users by using semantic-based models, (7) it must formulate and make available recommendations for registered users at the edge of the internet by using a rule-based approach to be applied on monitored data, (8) it must implement a service to be used for sending notifications to users via Telegram BOT, and (9) it must be capable of creating a sustainable, open, royalty-free, standard-based, and implementation-driven ecosystem. The resulting ecosystem aims to accelerate smart solution implementation in various domains, including healthcare and fitness, ensuring a standard-based approach.

Some additional information about this last requirements list is provided by following the same presentation order, except for requirements #6 and #7. By adopting a semantically-enabled architecture of an inter-domain RS, *App4Health* can use domain and extra-domain ontologies to formulate best practices and suggestions for the wellness and healthiness of its users. Indeed, semantics technologies can natively and smoothly support cross-domain relationships between respective concepts. In this way, even requirement #6 can be met at the same time. As its previous version, *App4Health* can

support a wide range of supported devices alongside domain and contextual data, being based on IoT and MCS technologies and adopting a context broker solution. In this way, the platform can refer to a wider informative domain which, in this case, is obtained by integrating data corresponding to the user's physiological state and the corresponding contextual conditions. Moreover, being compliant with the ECC, the *App4Health* can provide data preprocessing capabilities at the edge. Possible preprocessing operations include data storage, data enrichment and data ingestion, beyond the mere data preprocessing. These operations occur at both the edge and fog level, also because they are necessary to support rule-based recommendation formulation. In this way, even requirement #7 can be met at the same time. The middleware redevelopment represents one of the two requirements corresponding to specific component implementation. This requirement aims to adopt a context broker solution (instead of a message broker one) for interfacing the edge devices and to implement a data lake instance for supporting big data management spanning from the different levels of the ECC. In this way, it becomes possible to fragment and decentralize data management at the times and stages where it is most correct and suitable for them to take place. Even the persisting solutions to consider in this context are differentiated by adopting database solutions for managing structured, semi-structured and unstructured data. In this regard, the most relevant innovation is to adopt a context broker solution to replace the message broker component of the previous architecture version. Notification management represents the second of two requirements corresponding to specific component implementation. The notification management component is delegated to retrieve the suggestions/best practices formulated by the platform following a semantic- or rule-based approach and forward them to interested users through specific channels made available by appropriate social networking solutions. The target users to be considered for the notification forwarding are promptly and specifically selected by the platform essentially using their geographical position. Every hardware/software design and implementation choice within the project has been suitably evaluated and verified by first considering the compliance with the previous version of the architecture. The main reason for this stringent requirement is represented by the fundamental aim to adopt models and implementations that are already considered as the most mature, standardized and open possible. In addition to avoiding the possibility of unpleasant vendor lock-in situations, it is well known that only these requirements can foster and accelerate the creation of an ecosystem that needs to be sustainable, open, royalty-free, standard-based and implementation-driven. Therefore, the resulting ecosystem aims at favoring and accelerating the implementation of smart solutions, even in domains that could completely differ from the healthcare/fitness context. Doing so, also the last requirement can be met for the *App4Health* platform design and implementation.

3.2 | Architecture design

After identifying the RS requirements, it is possible to introduce the *App4Health*'s logical architecture that is shown in Figure 2. Such an architecture provides two tiers, namely edge/fog tier and cloud tier. Each of these tiers is then further structured into layers.

Before introducing the layers and components for each tier, it is worthwhile to provide some details about the cross-tier component called semantic service interoperability layer. The semantic service interoperability layer is a logical architecture that facilitates interoperability among services exposed by service, data lake, and IoT layers. It provides transversal functionalities that can be utilized by other layers, acting as a bridge between the services and the data they collect and share.

The edge/fog tier includes the IoT layer and part of the data lake layer.

The IoT layer represents the interface with the field sensors, the MCS devices and the other smart edge devices. Field sensors are used for detecting and monitoring environmental and contextual data. MCS devices are used for detecting and monitoring physiological data and to provide best practices and suggestions to users. Smart edge devices provide computational and storage capabilities for supporting data preprocessing, enrichment and ingestion. The context broker and the Telegram BOT sender represent the core components to data management and user interaction, respectively. These two components overlap the IoT layer and the data lake layer.

The data lake layer in *App4Health* is a logical architecture that overlaps the edge/fog and cloud tiers, as it stores and manages data from both tiers. Data management within this layer is performed by the same context broker, except for data sharing and exchanging. Indeed, the next version of the *App4Health* platform is planned to incorporate features for data sharing and exchanging, potentially using i4Trust or its alternatives.

The cloud tier consists of the semantic layer, the service layer and the interaction layer, beyond the remaining part of the data lake layer. The semantic layer includes the sets of both embedded ontologies and inference rules to be used for formulating the best practices/suggestions the platform must provide. The service layer provides the services set that

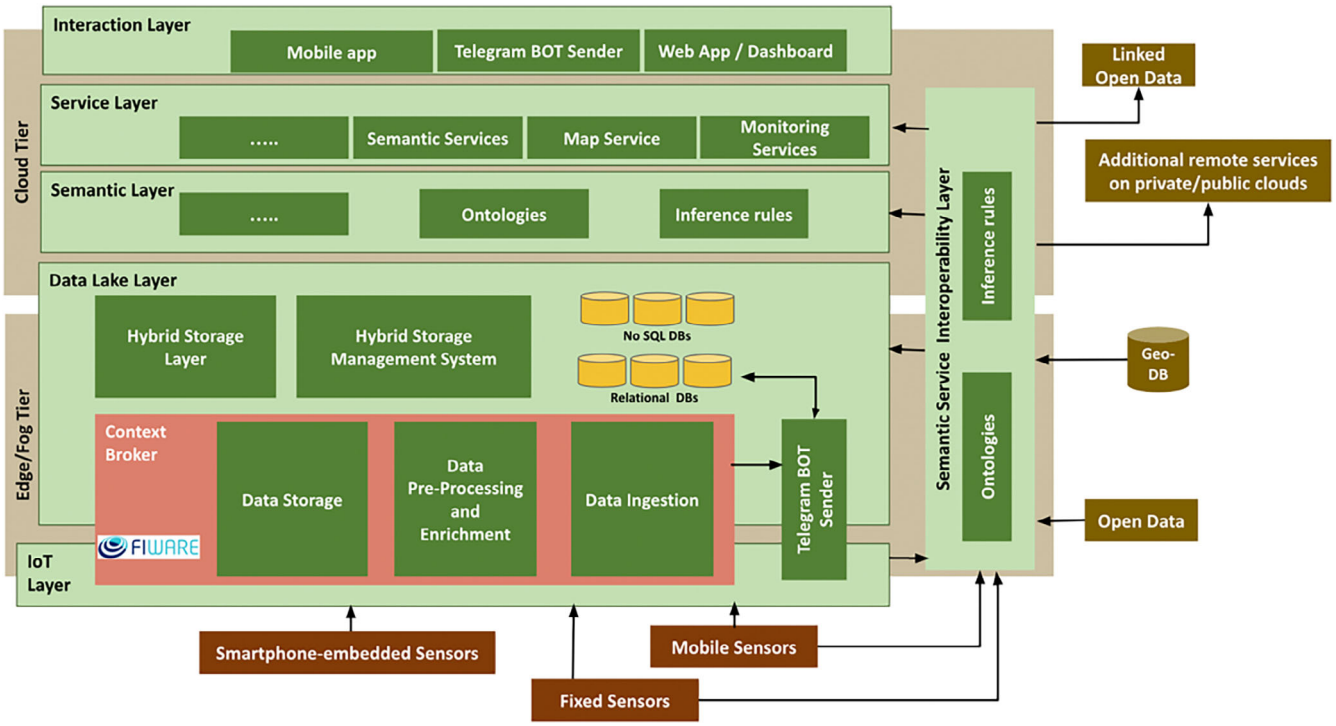

FIGURE 2 Overview of the *App4Health* platform architecture.

abstracts the semantic interoperability layer and that implements the same services it exposes. In addition, this layer implements an integration point to interact with maps and to support the monitoring activities. The interaction layer is the upper software layer of the *App4Health* architecture, responsible for user interaction aspects and supporting services and resources provided by the underlying layers for its own functioning.

Remembering the previous architecture, it can be stated that this complies with the osmotic computing paradigm by providing an abstract level between mobile/IoT devices and a cloud platform. Indeed, the abstract level represents the architectural middleware that enables opportunistic filtering for improving the data processing flow, also by considering additional metadata. The architectural middleware integrates data coming from mobile/IoT devices, specifically deployed in urban contexts using the osmotic computing paradigm. The middleware essentially corresponds to the data lake layer of the *App4Health* architecture. So, the *App4Health*'s data lake layer now embeds a context broker instance (i.e., Fiware Orion context broker) that is delegated to support the big data management. The main peculiarities of Fiware's context broker were already introduced in Section 2.4 and further discussed in Section 4.1.1.

By proceeding in this way, the *App4Health* architecture has been enhanced with respect to the previous one by implementing a solid and powerful infrastructure for data management. Beyond supporting big data management, the context broker instance offers features like tolerable latency, adequate queue size, scalability, and better performances, but mostly can support rule-based subscriptions on monitored data. Furthermore, an additional twofold result can be achieved by introducing a context broker instance within the *App4Health* architecture. On the one hand, the range of technologies and protocols used by the edge devices can be wider and complemented. On the other hand, the resulting infrastructure becomes particularly flexible and ready to integrate and interface the range of additional data management software components, including third-party solutions.

Another innovative aspect of the new release of the *App4Health*'s architecture with respect to the previous one is related to the semantic service interoperability layer component. It is a cross-tier component that extends from the edge/fog tier to the cloud tier and that provides the interoperability services using semantics. The semantic service interoperability layer makes available a semantic and multi-layered description of services, along with their actual compositions at a higher level. Doing so, the semantic service interoperability layer can provide a static description of the services, but also a dynamic definition of the corresponding workflow. With respect to the edge/fog tier, the semantic service interoperability layer can represent a further access point for the fixed and mobile sensors towards the semantic services it offers. Indeed, it can provide an integrated, shared and queryable representation of such services and compositions

thereof. On the contrary, from the cloud tier, the semantic service interoperability layer essentially interfaces the services of the *App4Health*'s cloud tier. However, for scalability sake, it can still interface remote services made available by additional private/public clouds. Further detailed information about the semantic service interoperability layer is present in Section 4.3. Finally, a notification service has been added to the architecture that operates at both tiers, edge/fog and cloud. The notification service corresponds to the Telegram BOT sender component. As the name suggests, such a component is delegated to best practice/suggestion management using the Telegram BOT technology. The reason for choosing Telegram BOT APIs resides in their prerogative to be a very widespread and adequately mature open technology with respect to other possible alternatives.

A detailed discussion of each tier, layer, and component the architecture of the *App4Health* platform includes is provided in the following sections.

4 | ARCHITECTURE IN DETAILS

The architecture in the large proposed in Figure 2 is detailed in the present section. For each tier, layer, and component the architecture of the *App4Health* platform includes a detailed description is provided.

4.1 | Edge/fog tier

Being part of an osmotic-oriented architecture, the edge/fog tier plays a crucial role in terms of features and services that it can offer. Being compliant with the osmotic specifications,[112] the edge/fog tier of the *App4Health* architecture is based on both the opportunistic computing and the concept of microelement. Beyond being lightweight, easy to move, and functional to a unique and well-defined purpose, a microelement can correspond to a passive or active unit which is respectively called microdata or microservice. The edge/fog tier aims at creating a software layer between mobile or IoT devices and the cloud tier. It enables opportunistic filtering and uses contextual data for improving the data processing flow.[32] The edge/fog tier overlaps on both the IoT layer and the data lake layer.

According to the opportunistic computing and ECC principles, the IoT layer provides interfaces with heterogeneous devices. Beyond sensors, the IoT layer includes edge devices that provide a minimum storage and computing capabilities suitable for specific tasks (e.g., data preprocessing, data filtering, etc.). Examples of computer-on-module solutions, such as Raspberry Pi or Arduino, fit to the purpose perfectly. This prerogative makes it possible for them to offer some basic value-added services at near real-time and with no dependency on the cloud support. According to osmotic computing, the target of the services a edge device offers may be any stakeholder, but also any other device that needs support in performing a task. Such a support may consist of storage and/or computing resources. Considering the wide range of data sources the IoT layer has to manage, the edge/fog tier has to implement a seamless integrated tier between the IoT layer and the data lake layer. The fog tier also provides load balancing capabilities in order to offer value-added services at the edge level and to reduce dependance and latency from the cloud tier. Specific aspects of the IoT layer and data lake layer are faced in detail in the following.

4.1.1 | IoT layer

The IoT layer constitutes the software layer that aims at abstracting the interaction with heterogeneous devices ranging from mobile devices to fixed monitoring stations, from surveillance equipment to personal-area-network-dependent devices. In particular, the monitoring activity can be accomplished by using environmental protection stationary equipment, mobile sensors mounted on the rooftops of local police vehicles or urban buses, or even smartphones and tablets with embedded/pluggable sensors (e.g., the HabitatMap AirBeam, the AirCare device) owned by citizens, in compliance with the MCS paradigm. These devices generate heterogeneous data streams according to the numerous parameters they monitor. In our case, the monitored parameters correspond to air quality (TVOC,* CO_2,† PM10,‡ and PM2.5§),

*Total volatile organic compounds concentration.

†Carbon dioxide.

‡Coarse particulate matter.

§Fine particulate matter.

environmental conditions (temperature, relative humidity, ambient light, atmospheric pressure, sound pressure level), and electromagnetic emission monitoring (both at high frequencies and low frequencies). The Aircare PRO[113] is the chosen IoT device we have used for monitoring and detection, essentially because it can be used for indoor and outdoor deployment. In the latter case, it is enough to have the foresight to place the device outdoors with protective shielding against wind and rain. Similar devices are typically used for ambient monitoring and complementing MCS campaigns as they can be placed in stable locations for longer periods than smartphones or tablets. The prerogative of Aircare PRO devices to be employed for both indoor and outdoor deployment represents the most relevant key factor in choosing them as our reference solution, given also the use cases we have to manage.

Being compliant with the osmotic architecture, this layer is capable of providing storage and computing resources closer to the edge of the internet. Data collected by the IoT layer is streamed to the data lake layer in order to populate the data lake. Albeit with a different purpose and level of detail, the IoT layer and the data lake layer can perform the same operations on data collected by sensors. These operations include data storage, data ingestion, data preprocessing and data enrichment. Finally, the IoT layer provides a specific integration point to the semantic service interoperability layer in order to access the services made available by data lake, semantic, and service layers. The software components that the proposed architecture provides at the IoT layer are the context broker and the Telegram BOT sender. A discussion regarding these software components is proposed below.

Context broker: To perform the data management within the IoT and data lake layers, we adopt a software component that belongs to the context broker category. Indeed, if compared with the previous version of the architecture, the most important innovation of the present tier is related to the presence of a context broker instance. The solution we have adequately selected for the context broker is Fiware Orion, which now performs the functions that were previously performed by the Kafka message broker. Fiware is an independent open community that offers a set of standard *APIs* by making available a complete and modular IoT platform back-end. Such a back-end makes it possible to Reference 114: (1) implement the interface with physical devices, by supporting APIs and protocols for communicating with a wide range of standard or proprietary devices, (2) adopt a Fiware *Next Generation Service Interface* (*NGSI*) broker for generating a context entity for each physical device, and (3) supply IoT device management features. Moreover, the context broker provides a local storage of the collected data by referring to a MongoDB database instance.

A minimal Fiware-based IoT architecture consists of both IoT Agents and IoT Agent Managers. While an IoT Agent represents a software module that can support various protocols, including Ultralight 2.0, JSON, or LWM2M, an IoT Agent Manager is an optional module used for configuring, operating and monitoring the reference IoT Agents. The prerequisites and purposes of the so-called Fiware project are essentially concentrated in choosing the Fiware core component as the reference implementation for the context broker. In this regard, the Fiware project mission is aligned with the mission of the present project. Indeed, the aim of this project to implement an architecture capable of creating an ecosystem that is sustainable, open, royalty-free, standard-based, and implementation-driven coincides with the mission of the European project Fiware. The additional reasons why we chose Fiware Orion as the reference context broker instance are the following: (1) it makes available a complete suite of additional GEs offering big data management functionalities, (2) it can support a wide range of communication protocols, and (3) it provides ample integration scope even with third-party solutions. Beyond being one of the possible context broker implementations that Fiware makes available, Orion represents the core mandatory component of a "powered by Fiware" application. To implement a complete big data platform, it is just then needed to choose and integrate the additional Fiware GEs to the reference context broker. It is important to underline that the features the context broker offers are used by both IoT and data lake layers.

Telegram BOT sender: The Telegram BOT sender is the platform communication channel that provides users with recommendations formulated by the *App4Health* platform using both the semantic layer and the context broker. For this reason, another instance of this component is available at the interaction layer of the cloud tier. It opted for this design choice to eventually provide best practices/suggestions even in near real-time and at the edge of the internet. The resulting recommendations formulated by both the semantic layer and the context broker are then stored into a relational database, that the Telegram BOT sender accesses to retrieve and forward them to the interested users using the reference Telegram BOT of the *App4Health* platform.

4.1.2 | Data lake layer

As the name suggests, the present layer aims at creating the condition to implement a data lake. A data lake can be defined as "a flexible, scalable data storage and management system, which ingests and stores raw data from heterogeneous

sources in their original format, and provides query processing and data analytics in an on-the-fly manner."[115] Essentially, a data lake implements a big data repository, which must support raw data storing, along with a corresponding set of management features, which is based on metadata descriptions.[116–119] Having to manage big data, a data lake must handle and store large and speedy amounts of unstructured data before processing them as quickly, in order to derive possible insights. For these purposes, a data lake allows to make immediately available collected data and to adopt dynamic techniques for the necessary data analysis.[120]

Data lake principles are the opposite of those exhibited by traditional data warehouse management. Indeed, a data warehouse can: (1) manage highly structured data, (2) adopt pre-build static data analysis techniques, and (3) offer support to slowly changing data. Recently, the emerging and pressing needs towards data sharing and exchanging within industrial contexts have given rise to concerns about the data sovereignty concept, especially when different and independent data lakes are involved. These same needs have contributed to the re-evaluation of the concept of data space, which can be apparently equated with the data lake concept. Essentially, a data space can represent an extension of a data lake implementation that is capable of supporting sovereign inter-organizational cooperation.

Since 2015, the International Data Space Association[121] has issued specifications on a reference architecture for data spaces, which are mostly borrowed from the GAIA-X project,[122] a central component of the Data Strategy of the European Union. More specifically, GAIA-X aims at building a "high-performance, competitive, secure and trustworthy data infrastructure based on European values."[122]

Observing the *App4Health*'s logical architecture, it can be noticed that the data lake layer overlaps the edge/fog and the cloud tiers, essentially because it must store and manage data produced by both those tiers. Except for data sharing and exchanging, the data management within the data lake layer is supported by the same instance of context broker we have already mentioned during the IoT layer discussion. The context broker represents a microservice subsystem that allows ingestion and storage collected data. At this stage, the context broker instance allows to meet the requirements related to big data management. Instead, the features needed to support data sharing and exchanging are already planned for the next version of the *App4Health* platform by adopting i4Trust[123] or one of its possible alternatives. In this way, the data lake implemented by the present version of the platform will also meet the additional requirements that are necessary to become a data space. Finally, the data lake layer provides a specific integration point to the semantic service interoperability layer for making available the accesses to the services it offers.

The software components that the proposed architecture provides at the data lake layer are the hybrid storage layer (HSL), the hybrid storage management system (HSMS), the databases set and the Telegram BOT sender. A discussion regarding these software components is proposed below, except for Telegram BOT sender which was already introduced in Section 4.1.1.

Hybrid storage layer: The HSL is a virtual database. It interfaces heterogeneous data sources and enables the corresponding data preparation pipelines. These preparation pipelines provide to organize, filter and annotate collected data, before storing them in opportunely storage solutions. For implementing the *App4Health*, the HSL we have adopted is Teiid[124] developed by JBoss community,[¶] which enables the integration of multiple database sources. This solution has been confirmed essentially because it is the HSL implementation that was already adopted in the previous version of the *App4Health* platform.

Hybrid storage management system: It is responsible for managing the reference heterogeneous databases. The HSMS component provides two modules related to both the HSL Admin component and the HSL Journal. Among other things, the latter module manages storage subsystem list, user profiles, system logs, and data growth estimation.

Databases set: According to the proposed logical architecture, the data lake layer insists on some internal databases and two additional external data sources. The internal databases correspond to some instances of both NoSQL and relational databases where opportunely storing collected data. In general, NoSQL databases are used to store raw data, while relational databases store data after they have been processed, enriched and ingested. The two additional external data sources, instead, consist of a geographical database and some open datasets, respectively. These additional data sources are particularly helpful in preprocessing and enriching the raw data. The database implementations we have used for implementing the *App4Health* platform are MongoDB[125] and PostgreSQL[126] as NoSQL and relational databases. Beyond providing open versions, MongoDB and PostgreSQL represent mature database solutions that are well-supported by the other software components of the *App4Health* architecture. Moreover, MongoDB represents the reference NoSQL

[¶]https://www.jboss.org/.

database used by the context broker. Finally, MongoDB and PostgreSQL provide a fervent user community that can represent a useful support and help when necessary.

4.2 | Cloud tier

Except for the semantic and inference aspects, the cloud tier essentially corresponds to the equivalent tier of a classical cloud-based IoT architecture. Beyond including part of the features the data lake layer offers, the cloud tier composition provides three layers (i.e., semantic layer, service layer, and interaction layer).

According to the previous platform architecture,[32] Apache Spark[127] keeps being the reference big data processing solution for implementing the cloud tier. Apache Spark was chosen as it is not only reliable and capable of real-time processing, but it is also delegated to perform complex data aggregation, computation, and processing, starting from the data collected by IoT devices (especially from data already preprocessed by fog devices). The present tier insists on a MongoDB database that is made available by the data lake layer and that is used for storing semistructured data. In addition to the previous platform version, this tier now offers new semantic aspects which have required some specific architectural adjustments.

The description in detail of each layer belonging to the cloud tier is provided in the following.

4.2.1 | Semantic layer

In order to properly describe the cloud services and their compositions, we have referred to the semantic representation described in Reference 128, which consists of a multi-layered description of services, along with their exposed operations and parameters as a foundation, and actual compositions at a higher level. Furthermore, a static description of such services and a dynamic definition of their workflow are provided. In particular, the latter is expressed in terms of *OWL-S*.[129]

The semantic representation, shown in Figure 3 consists of four levels, which can be further extended to provide definitions of elements other than cloud services. The bottom layer consists of the **Parameters Level**, in which semantic concepts for the representation of services' parameters are defined. At this level, concepts are used to understand what a specific parameter represents, and their definition depends on the cloud service they are used in. A *Parameter* has: a data type, a default value, a collection of allowed values and a generic description. A *Parameter* also partakes in the object properties *HasInput**HasOutput* as a *Range*, in order to be connected to the specific *Operations* exposed by the cloud service. Indeed, the ***Operations*** level provides a definition of operations\methods exposed by services. These are necessary to build workflows in *OWL-S*. An *Operation* is the domain of the object properties *HasInput**HasOutput* connecting them to *Parameters*, and it is the *Range* of the *HasOperation* object property, connecting it to the exposing service. The **Service** level represents the most complex of the layers, as it includes all services' definitions. It can be roughly divided into three parts:

- *AgnosticService*, a generic class used to represent Agnostic services, that is generic definitions that are not related to a specific provider, but that can be used as placeholders for functionalities and requirements.
- *ProviderService*, used to represent all services belonging to a specific provider. Individuals of this class will be connected to actual exposed operations, through the HasOperation object property.
- *ServiceCategory* is a fundamental class that is used to build an internal taxonomy of cloud services. This class is built as a hierarchical structure that describes several functionalities which can be exposed by services. As an instance, the Storage subclass holds references to storage services, and it is further refined through SQL and No-SQL subclasses. Composed services can belong to several categories, inheriting the possibility to use specific object properties.

The service level has its own properties. The *Requires* property describes the dependency existing between two services (either simple or composed). The *HasOperation* object property, connects the service, here in the domain, to the corresponding operation. The *HasCharacteristic* data property is the root of a more complex hierarchy that allows the definition of several characteristics for the service. As an instance, a store service is characterized by a *HasDimension* property, expressing its volume.

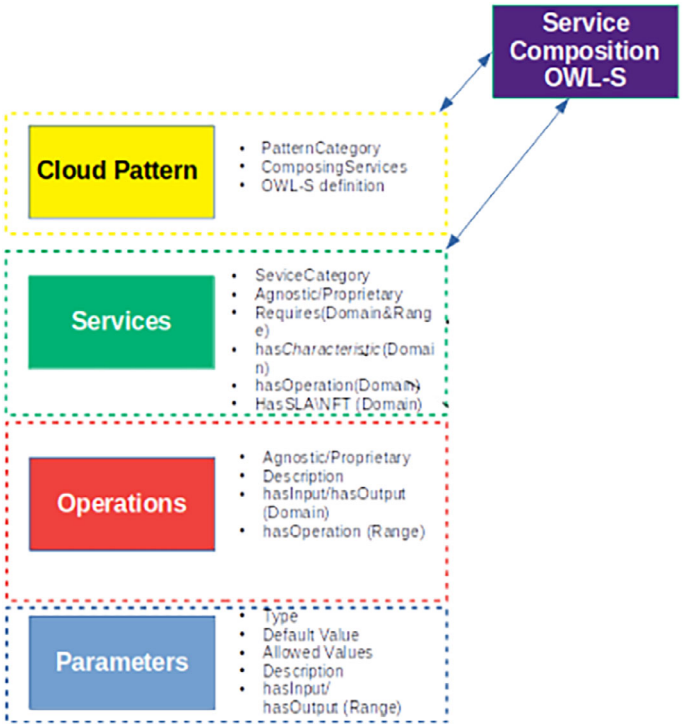

FIGURE 3 Overview of the semantic layer used to represent cloud services.

The cloud pattern level is at the top of the service representation. It includes the representation of composite services, as they are created by a service provider or automatically by a service composer. Here, a cloud pattern represents a set of existing services, combined together to offer a new set of functionalities, that are exposed as a new service or that can be used for further compositions. A *PatternCategory* class identifies the category to which each pattern belongs, making it possible to better search for an existing solution according to the required functionalities. The categories can overlap with the ones defined in the service layer. The semantic components that the proposed architecture provides at the semantic layer are two sets of inference rules and ontologies to be used for formulating best practices/suggestions. A discussion regarding these software components is proposed below.

- Inference rules: They are crucial for extracting new knowledge from populated ontologies and implementing specific functionalities. The inference rules can include multiple merged ontologies and provide additional integration. They are used by a semantic engine, which is essential for analyzing ontologies, checking consistency over time, and enacting the same inference rules. The semantic engine executes semantic queries on ontologies, providing an integrated view of the knowledge base.
- Ontologies: They represent the main component of this layer, which enables interoperability by defining common vocabularies for data and services, and automatically enabling matchmaking capabilities. These ontologies can be combined to integrate various aspects of interest, such as health and fitness, and Web API descriptions for communication, but these are just an example of the possible domains they can support.

The most relevant ontologies used for implementing the semantic layer are discussed below.

Semantic sensors Web API description

Available smart sensors used in IoT applications generally expose APIs that can be remotely accessed to extract the relevant information captured by the physical sensors. However, such APIs are generally extremely variable, often representing similar data and operations with different formats and terminologies. Also, sometimes the sensors are locked, that is the actual programming interfaces are not available, if not at a very low level.

The semantic sensors Web API description (SSWAD) ontology has the specific objective to provide a common, shared vocabulary for the description and definition of sensors' APIs, that can be used to represent input and output variables and operations of sensors by applying a completely agnostic and vendor independent approach. Based on the semantic sensor network ontology (SSNO), the SSWAD extends it by providing additional concepts, at an higher level, that provides a standard representation for sensors' API, which is independent from the specific device type available to the user. SSWAD provides a general, agnostic ontology that is usable to describe concepts referring to all kinds of sensors and devices. At the same time, the ontology can be extended by including other vendor-specific ontologies. In this way, sensor-specific APIs can be described and the relationships among parameters and calls (exposed by different device providers) can be easily assessed.

Figure 4 provides an overview of the SSWAD ontology. The hierarchical organization of the ontology allows for the representation of several Web API related concepts with different levels of complexity, and permits the connection with external ontologies, used for specific wearable devices. The core class is represented by **WebAPI**, collecting generic elements referring to sensor' APIs reachable through a web based connection. Such an API is then specialized by taking in consideration different possible aims of the used sensors: sleep tracking devices, for example, are better described by the **SleepAPI** class.

Another fundamental class is represented by **Parameters**, which is used to represent the input and output variables used to communicate with an API. The parameters class is specialized in specific sub-classes, describing different kinds of available parameters. Ontology provides support for the integration with vendor specific API ontologies, such as the FitBit API ontology described in the following.

The FitBit API ontology

The FitBit smart device's functionalities and operations are fully accessible through the internet, but specific API calls are needed to actually call them remotely. Since there are several requests that can be done, and each of them needs to be separately represented in the ontology, here a specific example will be used as a reference. Let's suppose we want to acquire the value of the heart-rate of a user, within a specific time span. In order to do so, we need to build a simple call like:

GET/1/user/[user-id]/activities/heart/date/[date]/[period].json

This is a basic RESTful request, where the variable elements are put in square brackets. In particular:

- User-id is the encoded ID of the user.
- Date is a reference date, in the format yyyy-MM-dd. The keyword "today" can be used.
- Period represents the time span to be considered. Values that can be associated with this variable are: 1d (one day), 7d (seven days), 30d (30 days), 1w (one week), 1m (one month).

Just from the observation of these few variables, it is clear that the API puts some kind of limitation to the possible values that can be used, especially regarding "period." All such information are translated into a device specific ontology, namely the FitBit ontology, which is strongly connected to the SSWAD ontology: indeed, many classes of the FitBit ontology directly extend, or refer to, classes in the SSWAD ontology. As an instance, the GET Restful request shown before, is represented as an individual of the FitBit ontology, belonging to the Get_Heart_Rate_Time_Series_by_Date class in the SSWAD ontology. The user-id, date and period variables are considered as subclasses of the URIArguments class, belonging to the FitBit ontology, as shown in Figure 5, and their individuals are used to refer to actual values used to build a RESTful request. Individuals of the URIArgument class are connected to the Get_Heart_Rate_Time_Series_by_Date class in the SSWAD ontology through the **hasURIArgument** object property. To define that the "period" variable needs to be limited, we use a restriction on a datatype property **hasPeriodValue** that connects the period class to the actual values associated with it during a request. The Protegè snapshot reported in Figure 6 shows how such a restriction is applied to the period class.

MCSO: The MCS ontology

The MCS ontology (MCSO) was designed to semantically model typical MCS. The MoCSO's modular structure proposed in Reference 130 was chosen as a foundation due to its consistent and import-efficient schemas, including SSN,

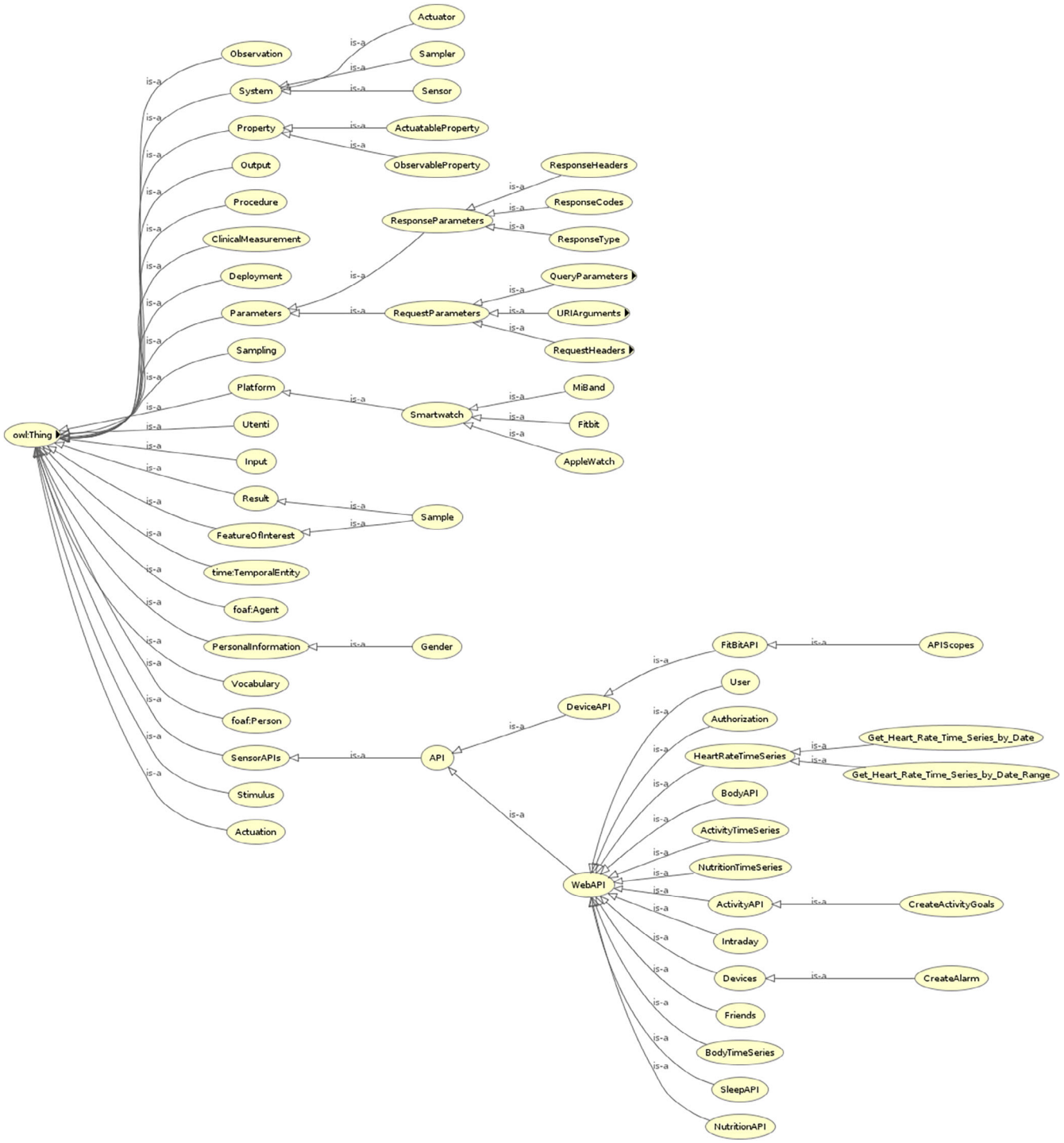

FIGURE 4 Overview of SSWAD ontology.

SSN-SYSTEM, and SOSA ontologies. The TIME ontology[131] was imported to manage timestamped sensor readings. Two more ontologies were included: the WGS84[132] and the Context (CXT) ontology.[133] WGS84 is used to specify spatial locations, while CXT is used to identify application scenarios for the MCS paradigm. Although CXT was initially created for ambient intelligence, it was selected because of its lightweight structure that can be easily aligned with some core entities from SSN/SOSA.

Moreover, MCSO required at least another ontology capable of describing units of measurement, since they are essential for every sensor measurement and equipment specification sheet. Several ontologies exist in this field,[134] including QUDT,[135,136] OM2,[137,138] and UO.[139,140] Although QUDT and OM are suggested integrations by SSN developers,[141] they

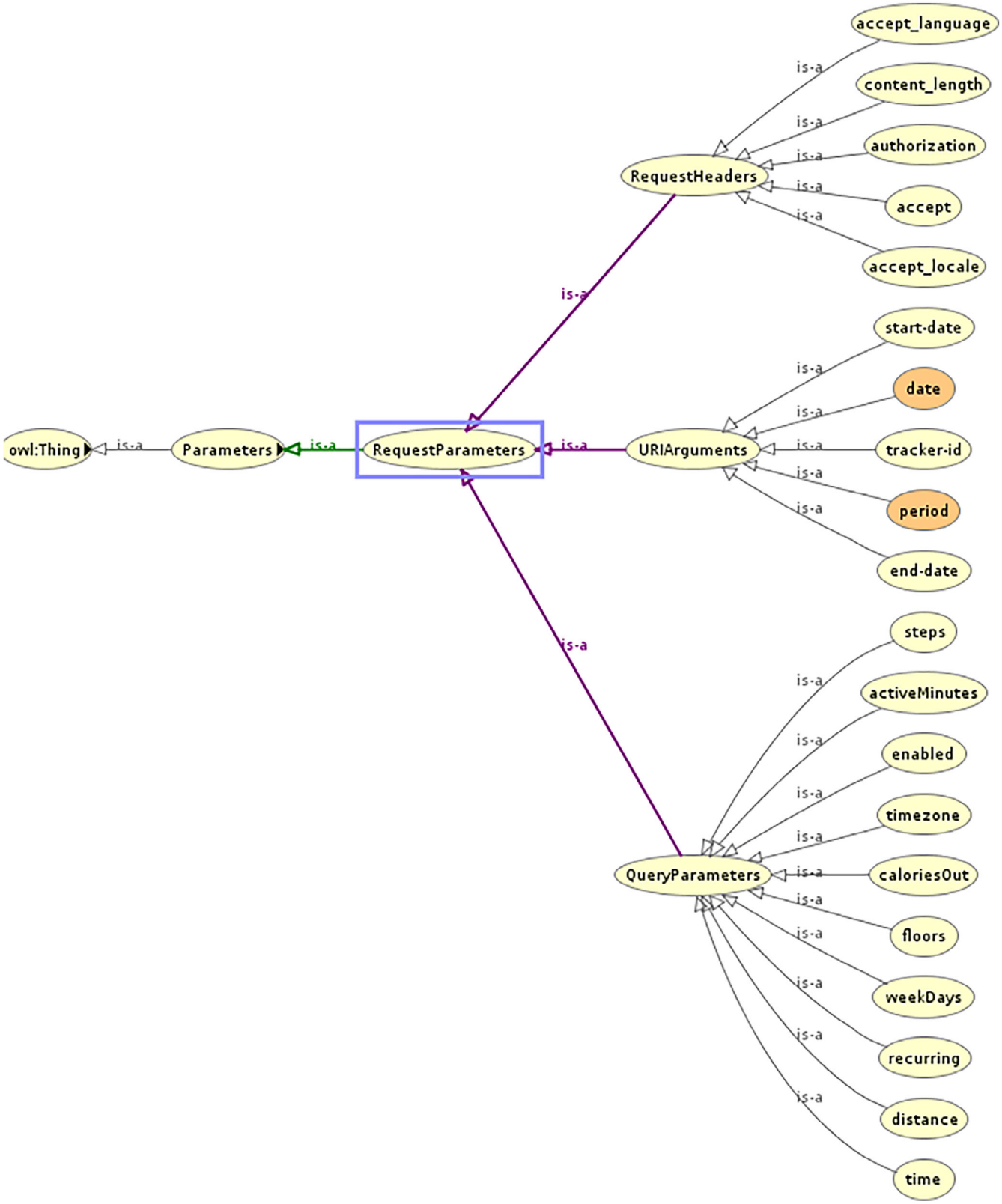

FIGURE 5 Overview of parameter class in the FitBit ontology.

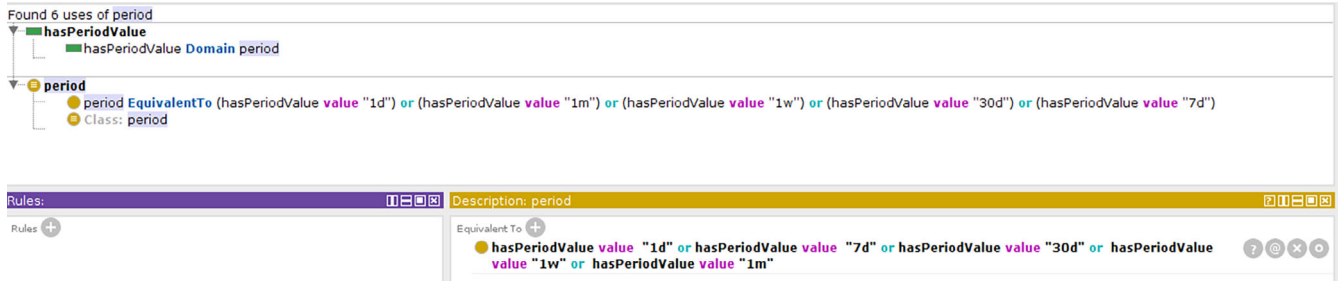

FIGURE 6 Restriction on the "period" class.

present issues when imported via their URI in ontology editors requiring OWL2 consistency due to the presence of some pure RDF/RDFS statements. Consequently, also in this case, UO was selected as a replacement for its lighter structure. Lastly, tasks and activities were modeled after COMCS ontology.[142]

To construct the MCSO ontology, we employed a two-pronged approach. First, we interviewed domain specialists in MCS and citizen science to obtain domain-specific design requirements. Second, we collected functional requirements from prior MCS endeavors conducted on a regional, national, and international scale, covering diverse monitoring scenarios such as noise, electromagnetic and air pollution.[143–145]

The ontology construction requirements were established as follows. First, the ontologies must support modeling hybrid MCS scenarios with sensor readings from both smartphones and fixed sensing stations, which is common when calibrating smartphones against fixed station measurements. Second, the ontologies must enable modeling scenarios in which users participate in MCS campaigns with their smartphone and later connect the same device to wearable sensors during their own fitness activities.

We utilized a loose coupling approach between the MCSO ontology modules and the SSN ontology by aligning key entities at the class level. Our top-level classes were selected and declared equivalent to corresponding entities in the SSN ontology. In some cases, a tighter coupling was adopted, such as our observable physical phenomena classes that were declared direct subclasses of the *Feature of interest* class in SSN. The loose coupling between MCSO and SSN resulted in 18% of equivalent classes and 6% of equivalent object properties. An additional 7% of classes in MCSO were declared direct subclasses of corresponding SSN classes. For any additional details about MCSO we refer the interested readers to Reference 108.

4.2.2 | Service layer

The service layer comprises services that facilitate the provisioning of processing outputs from underlying layers to the final layer (i.e., interaction layer), enabling users to interact with the entire system. The service layer provides the services set that abstracts the semantic layer and that implements the same services it exposes. In addition, this layer implements the business logic to interact with maps and to support the monitoring activities. Most of the implementations corresponding to these services have been reused, while the semantic services component has been fully designed and implemented for the *App4Health* platform. The semantic services component includes the business logic that we have developed to support the interaction with the semantic layer. The service layer provides a specific integration point with the cross-tier semantic service interoperability layer for making available the services it offers. The software components that the proposed architecture provides at the service layer are the following: semantic services, map service, and monitoring services. Java was adopted as the programming language for the implementation of the additional microservices developed for the latest version of the platform, in order to maintain an adequate level of coherence with the microservices developed previously. Moreover, Java also allowed us to cope with varying workloads and with the inherent dynamicity of those microservices that are highly concurrent and demand for significant scalability. A discussion regarding these software components is proposed below.

Semantic services: They abstract the entire semantic layer by exposing the services such a layer effectively implements and makes available to other architecture layers that need them.

Map service: It enables the interaction with the reference map system in order to support users in geographically navigating and consulting the user interface.

Monitoring services: They complement the monitoring activities performed at the edge/fog tier of the *App4Health* platform. In this way, it becomes possible to complete the data detection and monitoring process related to the data corresponding to the users' physiological state and to the contextual one as well.

4.2.3 | Interaction layer

The interaction layer consists of the upper software layer of the *App4Health* architecture that takes care of the user interaction aspects. For its own functioning, such a layer uses the supporting services and resources offered by the underlying layers. The software components that the proposed architecture provides at the interaction layer are the following: mobile app, Telegram BOT sender, and web app/dashboard. A discussion regarding these software components is proposed below, except for the Telegram BOT sender component which is one of the two instances the architecture includes and a detailed description was already reported in Section 4.1.1.

Mobile app: It represents the MCS means by which the user performs the monitoring and detection activities through embedded and/or plug-in sensors. As a result of these activities, the app can alert citizens in case of relevant situations and/or conditions that can safeguard or protect users during their outdoor and indoor fitness activities.

Web app/dashboard: It consists of a pool of dashboards capable of opportunely presenting and summarizing contents, also by differentiating them according to a user role-dependent approach.

Each of these components has been adjusted and extended to meet the requirements we have already detailed within Sections 3.1 and 3.2.

4.3 | Semantic service interoperability layer

The semantic service interoperability layer provides transversal functionalities that can be exploited by other layers, as shown in the reference logical architecture proposed in Figure 2. As the name suggests, the primary objective of this layer consists in enabling interoperability among services exposed by the service, data lake and IoT layers, acting as a bridge between the respective services and the corresponding data they collect and share. The input of such a layer is represented by three typologies of artifacts:

- **Ontologies** that are used to describe the characteristics of the services in a general, abstract and agnostic manner. The services' descriptions are used as a common background to enable interoperability, through the mapping of concepts, entities and definitions from one specific service domain to another. Ontologies included in this layer have to be considered upper ontologies, since they are not tied to a specific domain, but act as a general reference, a common background for other services' descriptions.
- **Inference rules** that are needed to identify similarities and equalities among described services. Inference rules define the conditions that determine if two services are naturally interoperable, and can therefore be used together without effort. They are also able to identify the specific transformations that are needed to implement such interoperability, when possible. These rules can be used to determine, given a specific function that needs to be implemented, which services can be automatically composed, or eventually adapted, to provide the required functionality.
- **SPARQL queries** are employed to retrieve information about services and IoT devices, facilitating their composition for achieving specific functionalities. Ontologies serve as the foundation of knowledge, while inference rules enable knowledge expansion based on codified logic. Meanwhile, SPARQL queries play a crucial role in collecting and filtering this knowledge to align with the requirements of the application.

The upper ontologies used within the semantic interoperability layer must take into account the semantic description related to each service provided by the same layer, along with external data sources. Open and linked open data are considered as possible sources of external information, since open datasets can be used to provide public description for each service and the corresponding characteristics, but also to obtain descriptions and definitions of functionalities, especially

related to IoT devices and services. The different combinations of these descriptions and definitions are extremely useful to define complex compositions of interoperable services.

5 | PLATFORM EVALUATION

The distributed and multilevel architecture we have proposed for the *App4Health* platform is capable of implementing a complete IoT data management system. By adopting the ECC approach, the resulting management in terms of data processing and storage becomes compliant with the fundamental characteristics of being decentralized and distributed along the various architectural layers. The main goal is to be able to create the conditions for the provision of proximity-value-added services and in near-/real-time. These services must be available very close to the places and at the times where the corresponding data are produced.

Bearing in mind the processing and storage capabilities that the edge devices must guarantee, it becomes clear that the corresponding major criticisms can essentially affect the hardware and software components that are located at the edge/fog tier. Starting from this assumption, it is crucial for the edge and fog software components to ensure constant and adequate levels of availability and reliability. Representing the direct interface with respect to the IoT data streams, the edge/fog tier must be able to tolerate load peaks, considering that it must potentially support the subsequent creation of value-added services at the right times and ways, beyond implementing additional activities like data storage, data preprocessing and enrichment, and data ingestion. The architectural software components of the edge/fog tier that we believe are most interested in this purpose are certainly the context broker and the Telegram BOT sender. Representing the main communication channel with the user, the latter component must guarantee an efficient and adequate level of notification service.

One of the main value-added services of the proposed architecture is precisely the notification to the interested user base of constant and continuous updates and/or suggestions about health best practices to be observed, based on the respective personal conditions. To be useful and effective for users, these updates must obviously be notified to the interested parties in the appropriate times and ways.

This section aims to analyze and verify the levels of availability and reliability of the context broker and the Telegram BOT sender through a stressing evaluation test, detailed below, which involves a comparative stress test of the aforementioned software components, which will be deployed on smart edge devices using different hardware equipment. The comparative stress test tries to identify what workloads the software components can tolerate in order to guarantee an acceptable level of service, according to the edge device on which they are deployed. We assumed that the Telegram BOT sender component is released both at the edge device level, and on a cloud infrastructure. In the first case we expect: the zeroing of latency times for the interaction between the edge and the cloud and a significant improvement in terms of availability and performance in providing value-added services. In order to further clarify, we propose a case study referred to a gym context in which an *App4Health* user is busy carrying out physical activity in one of the gym rooms. Both the gym rooms and the user are monitored through specific sensors and the related data is periodically transmitted to the edge/fog devices of the architecture envisaged by *App4Health*. By making available an instance of both context broker and the Telegram BOT sender for the case study, it becomes possible to provide proximity-value-added services as best practices/suggestions by essentially collecting, preprocessing and inferring user's data with the corresponding contextual one. For the aim of the case study, it is the Telegram BOT sender service at the edge/fog tier to be used for delivering these best-practices/suggestions. The rule-based formulation of best-practices/suggestions operated by the context broker is discussed in Section 5.3. While the semantics-based formulation of best-practices/suggestions has been detailed in two case studies included into a previous work[108] from the same research group.

5.1 | Testbed description

This subsection deals with the comparative stress test of the context broker and the Telegram BOT sender components, at the edge/fog tier. The performance degradation threshold of those components was identified assuming that they are deployed on edge-level devices, according to two configurations. The first configuration is labeled as C1SD (Configuration 1 Single Device) and consists of just a single edge device; the second configuration is labeled as C2CD (Configuration 2 Clustered Devices) and employs a cluster of five smart edge devices. Therefore, their processing power is considerably different.

5.1.1 | Edge hardware devices

In our testbed, we deployed the edge/fog software components on two different types of smart edge devices: (1) a single Raspberry Pi 3-Model B+; (2) a cluster of 5 Raspberry Pi 4. A brief description of these devices is given below, together with the related, most relevant technical specifications.

1. **C1SD configuration: Raspberry Pi 3-Model B+**—One of the most common edge-level single board computers (SBCs) is the Raspberry Pi.[146] It is a microcomputer available in various models that adopts a specific distribution of the Linux open-source operating system Raspbian. The device is powered by a micro-USB connector and has a slot to host an SD card where storing both the operating system and data. The decision to opt for the Raspberry Pi is also motivated by its characteristics of being a very popular, cheap, and flexible SBC that is based on an ARM architecture processor capable of supporting a complete Linux distribution. A possible alternative to the Raspberry Pi is Arduino,[147] although the latter has much lower computing power and storage, thus making the Raspberry Pi a more suitable device for the testbed. Such a device can provide adequate computing and storage capabilities, considering that it must interact with the IoT layer and it must opportunely support the context broker and the Telegram BOT sender components. The requirements that the Raspberry device must meet are the complete management of both the IoT and the Telegram BOT sender data communications. For the purposes of this use case, the Raspberry model we have used is the PI 3 Model B+, having a 64-bit Cortex-A53 (ARMv8) processor at 1.4 GHz, and 1 GB of RAM memory.
2. **C2CD configuration: Raspberry PI 4 Cluster**—For developing the *App4Health* platform assessment, we have used a desktop cluster or micro data center consisting of a cluster of 5 Raspberry PI 4 boards. For sure, it can represent a potential smart edge device, also by eventually opting among different hardware configurations. For the purposes of this testbed, the Raspberry model we have used is the PI 4, having an Athlon II X2 Dual Core 245 processor at 1.5 GHz, and 8 GB of RAM memory. Considering the resources that such a device can offer, it becomes possible to use it for running a wide variety of software components, including web servers, relational and NoSQL databases, but also container managers like Docker Swarm or Kubernetes.

5.1.2 | Edge software components

The testbed we have conducted for the comparative performance analysis of the *App4Health* platform implements a typical software architecture with two possible hardware and software configurations. Such an architecture consists of three software layers corresponding to the device simulators, the *App4Health* preprocessing, and the Telegram BOT clients. A brief introduction for each of these layers has been provided in the following.

1. *Device simulators*: it represents the Python module we have implemented to simulate the thread-based REST calls that the devices address to the Fiware Orion context broker exposed by the *App4Health* preprocessing layer. In order to stress the context broker software component, we have referred to typical REST calls that an IoT device uses to transmit air status and quality updates corresponding to the environment that the device itself monitors. These REST calls use the POST method and a JSON payload for data transmission. For stressing the Telegram BOT sender edge component, instead, we have opted to force it to perform a massive message sending to the preconfigured Telegram BOT. The message texts to send are randomly retrieved from a set of different messages that is stored into a relational database instance. By monitoring the overall response times for fully serving requests addressed to both the context broker and the Telegram BOT sender, it is possible to terminate the script execution as soon as the same response times reach a preconfigured maximum value.
2. *App4Health* preprocessing: considering that the *App4Health* platform was implemented as a Docker-based micro-service architecture, the *App4Health* preprocessing layer consists of the context broker and the Telegram BOT sender services of the same platform. Depending on both hardware and software characteristics of the device on which the platform is deployed, this layer can provide single or multiple instances of the context broker and the Telegram BOT sender services. Obviously, we have arranged the testbed providing a single instance or a variable number of instances for each of these services in correspondence of the C1SD and C2CD configurations, respectively. In the latter case, the device was equipped with a Docker Swarm-based cluster, with a single manager node and four worker nodes.
3. *Telegram BOT Clients*: they represent a set of mobile phones that receive the notifications formulated by the *App4Health* platform. They constitute the registered users' personal devices that are equipped with a Telegram

app to receive these notifications. To this aim, a registered user has to start the interaction with the *App4Health* Telegram BOT.

The testbed environment for performing the *App4Health* evaluation has been carried out at the DataLab laboratory of the Department of Engineering for Innovation of the University of Salento. In particular, for carrying out the comparative platform evaluation, it was necessary to organize a testbed with two different configurations, namely C1SD and C2CD. These configurations correspond to the deployment of the *App4Health* preprocessing software layer of the aforementioned architecture on a Raspberry PI 3 and on a Raspberry PI 4 Cluster device, respectively. The other two software layers of the architecture remain essentially the same for both testbed configurations. For implementing the *Device simulators* software layer, we have used a workstation that simulates data streams transmission generated by fixed and mobile sensors and smartphone embedded sensors. The data streams generation is simulated by executing a Python script that sends bursts of thread-based REST calls addressed to the context broker. Instead, for implementing the *Telegram BOT Clients* software layer, we have used 10 heterogeneous mobile devices equipped with a Telegram App. Each of these running instances of Telegram App was previously registered to the *App4Health* Telegram BOT. In this way, each of these devices can receive the notifications generated by the context broker and sent by the Telegram BOT sender.

C1SD configuration: Software deployment—For discussing the present configuration, we will focus on the *App4Health* preprocessing layer, because the two remaining software layers have been similarly implemented for both the testbed configurations. In this regard, the only difference between the two configurations is related to the three different running settings necessary to configure the execution of the Python script of the *Device simulators* software layer, related to: (1) the number of thread-based REST calls for the first burst, (2) the number of parallel REST calls to perform for the subsequent bursts, and (3) the maximum delay threshold in seconds for fully serving a request. In this case, the values we have used for the running settings are 100 for the REST calls number of the first burst, 400 as REST call number for the subsequent bursts, and 1 s as the maximum delay threshold respectively. These values are the result of some previous test experiments we have conducted aimed at identifying a possible optimal values combination for the running parameters, with respect to the resources of the device used, the relative response times and the yield of the representation of the performance trend. The same approach was adopted also for identifying the running parameters to be used for developing the second testbed configuration. The reference architecture for C1SD configuration is shown in (Figure 7). According to C1SD specifications, the Docker services the *App4Health* preprocessing layer must provide were deployed on a single Raspberry PI 3 Model B+ device. So, only one instance of the context broker and the Telegram BOT sender services is available for serving the REST calls bursts the *Device simulators* layer progressively generates. Each service includes the corresponding database services as well. While the context broker interfaces a MongoDB NoSQL instance, the Telegram BOT sender insists on a PostgreSQL relational database.

C2CD configuration: Software deployment—Once again, for discussing this configuration, we will focus on the *App4Health* preprocessing layer, because the two remaining software layers have been similarly implemented for both the configurations. The running settings necessary to configure the execution of the Python script of the *Device simulators* software layer remain the same, even in this case. Obviously, their value changes considering that the target hardware device, a Raspberry PI 4 Cluster device, has now more processing and storage capabilities. Indeed, the values we have used as running settings are: 100 for the REST calls number of the first burst, 4000 as REST call number for the subsequent bursts, and 1 second as the maximum delay threshold. The reference architecture for C2CD configuration is shown in (Figure 8). According to C2CD specifications, the Docker services the *App4Health* preprocessing layer must provide were deployed on the Raspberry PI 4 Cluster. Beyond Docker and Docker Compose, the running environment for the present configuration also includes an instance of Docker Swarm cluster. The Docker Swarm instance makes it possible to create a 5-node cluster, where each node corresponds to a Raspberry PI 4 of the clustered device. The main advantages in referring to a Swarm cluster are the following:

- Cluster creation and management: smart devices are integrated with Docker Engine and no additional orchestration software is needed.
- Decentralized design: it is possible to opt between the manager and worker roles for each node of the cluster at runtime.
- Scaling: this operation can be automatically performed by the swarm manager, or appropriately set at runtime by using specific custom settings. Each service can be scaled up or down.
- Service discovery: for each service of the swarm, the corresponding manager nodes provide for a unique DNS name and load balances running containers.

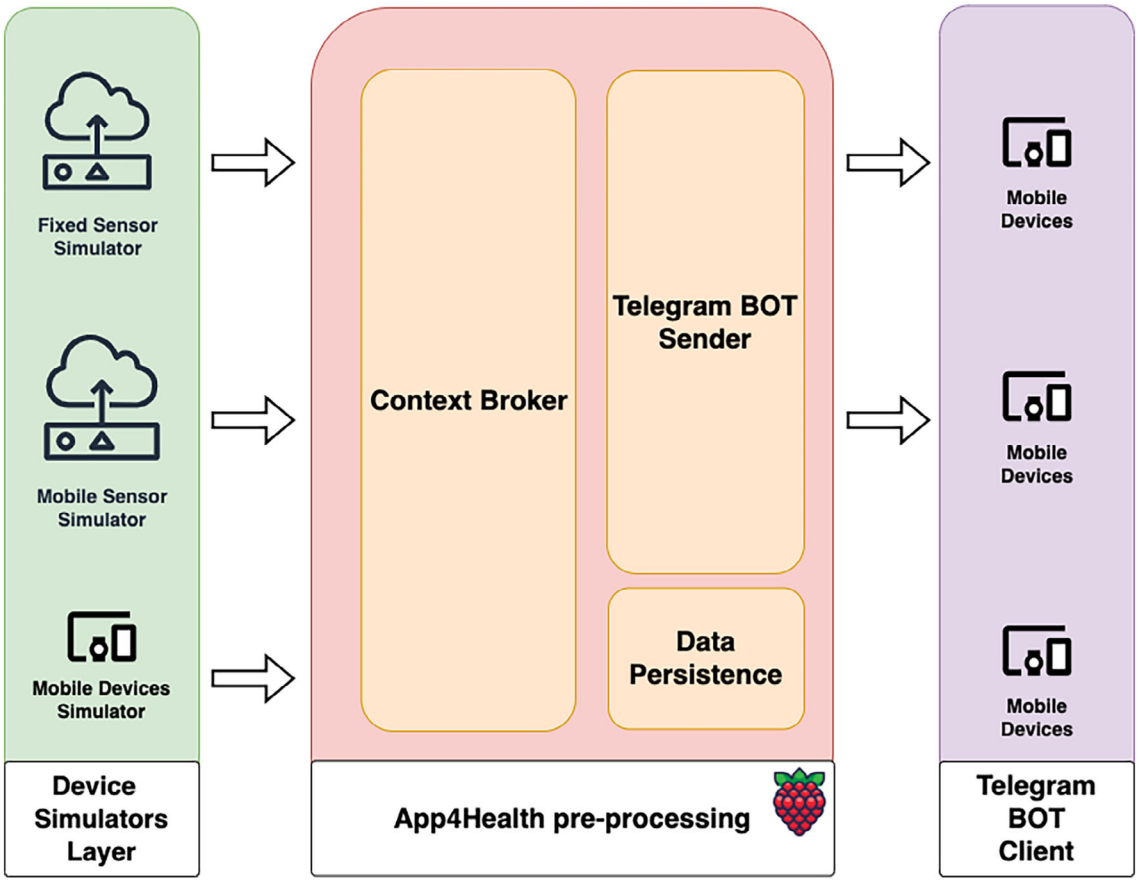

FIGURE 7 C1SD configuration: Software architecture.

- Secure by default: connections between the swarm nodes are secured by providing TLS mutual authentication and encryption.

Moreover, the reference cluster for C2CD configuration consists of one manager node and four worker nodes. For each node, we have deployed 2 instances of both the context broker and the Telegram BOT sender Docker services. Each service includes the corresponding database services as well. While the context broker interfaces a MongoDB NoSQL instance, the Telegram BOT sender insists on a PostgreSQL relational database. Overall, 10 instances are available for both the context broker and the Telegram BOT sender services. These instances are used for serving the REST calls bursts that the *Device simulators* layer progressively generates.

5.2 | Performance analysis

The overall results of the *App4Health* platform are shown in Figure 9, where the orange line represents C1SD configuration (consisting of a Raspberry PI 3 Model B+), the blue line identifies C2CD configuration (comprising a cluster of Raspberry PI 4), and the yellow line corresponds to the delay threshold we have set. Considering the line related to the C1SD configuration, it can be noted that it intersects the delay threshold line at the seventh burst cycle, in correspondence of around the 2400th REST call. As it was easy to imagine, the line corresponding to C1SD configuration carried out using a single Raspberry PI denotes a delay that follows a decidedly exponential trend, after a short fluctuating transient. Instead, considering the line related to C2CD configuration, it can be noted that this line intersects the delay threshold line at the seventh burst cycle, approximately at the 26,500th REST call. If compared against the C1SD configuration trend related to a single Raspberry PI, the line corresponding to the C2CD configuration denotes a delay trend that becomes exponential in a less sudden way. Indeed, once more after a short fluctuating transient, the delay trend remains essentially

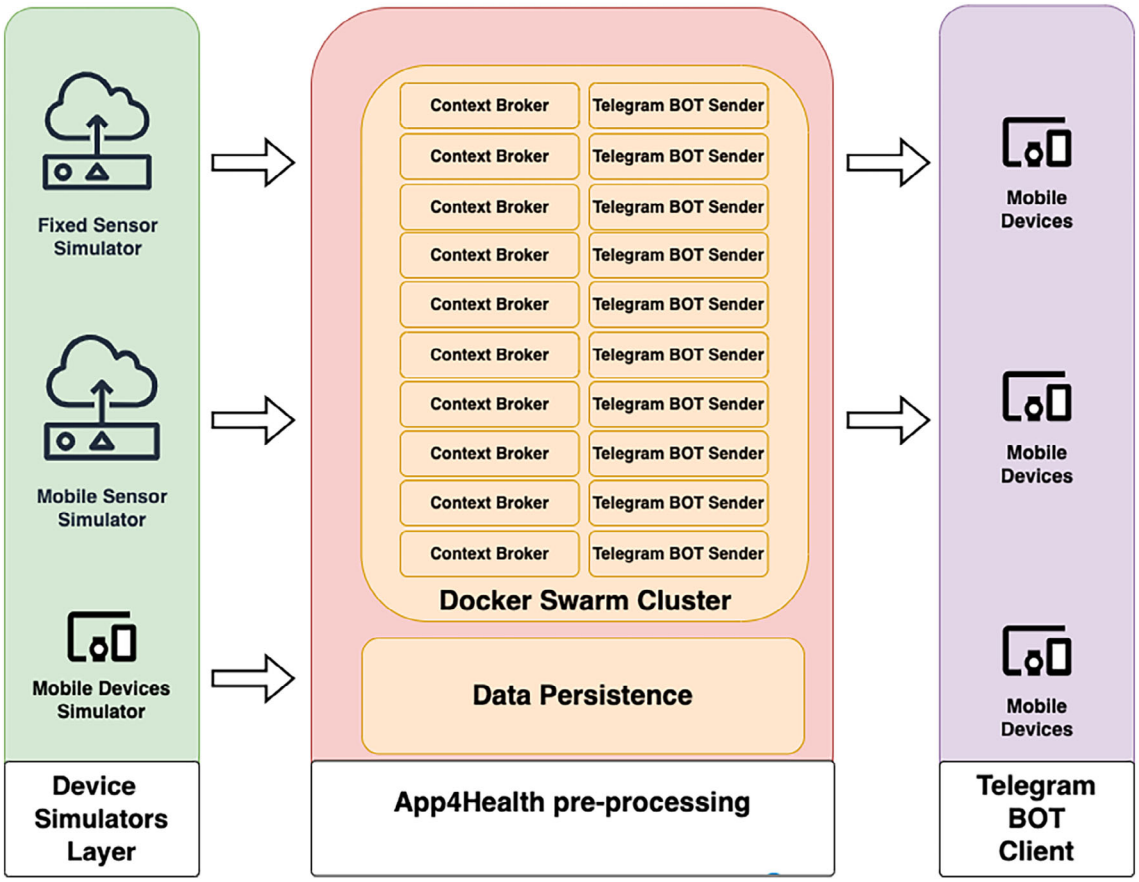

FIGURE 8 C2CD configuration: Software architecture.

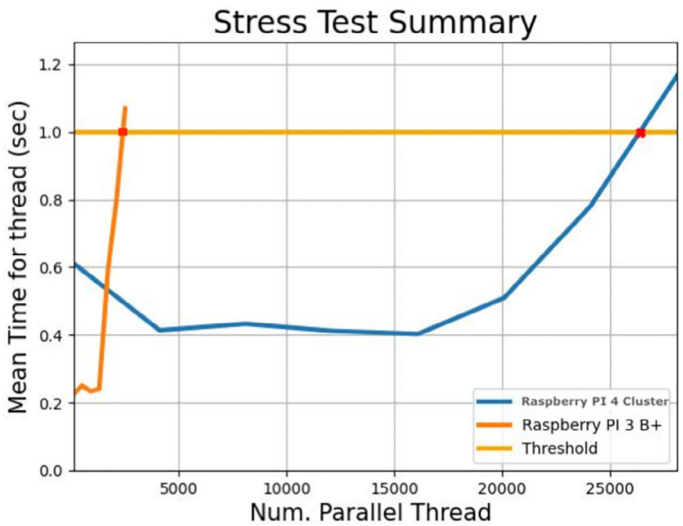

FIGURE 9 *App4Health* platform assessment result.

concentrated between 4 and 5 tenths of a second, up to about the end of the fifth burst of parallel REST calls. By deeply analyzing the comparative performance result corresponding to both the configurations, it is possible to state that the response times obtained by the examined *App4Health* services deployed on two different hardware devices and referring to different software solutions follow a linear trend as processing and storage capacity increases, at least up to a certain workload level. Considering the premises and criteria adopted when choosing the architectural components to be subjected to stress tests, the previous considerations can be for sure extended to the overall *App4Health* platform. Indeed, the context broker and the Telegram BOT sender software components represent the services whose performance has the greatest impact on the overall performance of the entire *App4Health* platform.

5.3 | Case study

For better illustrating how *App4Health* platform works and for further demonstrating the usefulness of its features, we propose a case study on a significant edge-fog scenario providing an added-value service. For the sake of simplicity, the case study is developed by limiting data collected by sensors with the aim to focus on how the *App4Health* platform works. Indeed, this limitation does not impact on the significance of the proposed case study.

The case study is related to a gym setting where an *App4Health* registered user is working out in one of the workout areas. Through specialized sensors, the user and the exercise rooms are both monitored, and the associated data are periodically sent to the edge/fog devices of the architecture envisioned by *App4Health*. Leveraging on both context broker and Telegram BOT sender edge services, it becomes feasible to offer proximity-value-added services as best practices or suggestions. This is done by basically gathering, preprocessing, and inferring user data and the corresponding relevant contextual data. Supposing a registered user wears sensors or a smart device, such as a FitBit, to monitor his/her physical activities, physical effort, and fatigue.

An additional sensor is then deployed in each gym room to detect nearby buildings' indoor environmental parameters (e.g., temperature and humidity). The *App4Health* platform uses the Telegram BOT channel to warn the user not to enter a specific indoor location due to adverse temperature conditions, potentially affecting his/her health status. In this regard, a significant contribution is provided by the Fiware Orion context broker component to implement the mechanism for eventually triggering the alert message for the user. Consequently, the context broker is aware of both the physiological status of each user inside the gym and the climatic conditions of each gym's room, being constantly updated on these data by the corresponding smart devices that are hereby deployed. For the aim of the present case study, we assumed that:

1. The given gym consists of three rooms.
2. Each room in the gym is equipped with a sensor to monitor at least the corresponding ambient temperature.
3. The *App4Health* registered user is equipped with a smart device to track at least the user's heartbeat rate and temperature, along with his/her current GPS position.
4. A room in the gym (Room B) has the air conditioning system that begins to malfunction, causing the relative temperature to drop.

For the sake of clarity, the discussion introduces the data models used by the *App4Health* application to synchronize the state of each relevant actors/assets present in the proposed scenario with the state of the corresponding digital entity. In the proposed scenario, the relevant actor is the *App4Health* registered user, whilst the relevant assets are the gym and the three corresponding rooms. All the data models provide four common attributes, namely id, type, name, and location. Id and type are used to uniquely identify an entity among the various ones managed by the context broker; whilst, name and location respectively represent the proper name and the GPS coordinates (expressed as GPS point) of the corresponding entity. These GPS coordinates are detected and transmitted by specific fixed or mobile sensors.

In this context, exact user's GPS tracking is not a stringent requirement, as the location is just used to locate gym rooms within a 500 m radius with inadvisable conditions given the user's physical state. So, inexact GPS positions can be adequately tolerated.

In this regard, is it planned as future work to adopt standard data models that are compliant with the NGSI-LD[148] format, but also an alternative approach to internal building GPS tracking. In particular, it is planned to match the user's position to the coordinates of the gym room in which it is actually located.

In the following, a short description for each JSON data model related to adopted actor/assets is provided.

```
      {
        "id": "User1",
        "type": "User",
        "heartbeat": 70,
        "temperature": 36,
        "location": {
            "type": "Point",
            "coordinates": [18.17244001, 40.35481001]
        },
        "name": "Lastname Firstname"
      }
```

Listing 1: *App4Health* registered user: Data model definition.

User Actor: JSON data model—Listing 1 shows the data model corresponding to the *App4Health* registered user. Beyond id, type, name, and location, additional attributes for the present data model are the complete name, heartbeat rate, and temperature of a given user. Heartbeat rate, temperature, and location attributes will be used by the context broker to check the user's physiological condition and to eventually individuate gym rooms with internal temperature that can cause health problems to the same user.

```
{
    "id": "Gym1",
    "type": "Gym",
    "rooms": ["Room1","Room2","Room3"],
    "location": {
        "type": "Point",
        "coordinates": [18.1724400, 40.3548100]
    },
    "name": "The Gym"
}
```

Listing 2: Gym: Data model definition.

Gym Asset: JSON data model—Listing 2 includes the data model related to the specific gym. Beyond id, type, name, and location, the only additional attribute for the present data model is the corresponding rooms collection. Such a collection includes the three ids of the available rooms at the considered gym.

```
{
    "id": "Room1",
     "type": "Room",
    "room\_temperature": 23,
    "location": {
        "type": "Point",
        "coordinates": [18.17244001, 40.35481001]
    },
    "name": "Room A"
}
{
    "id": "Room2",
    "type": "Room",
```

```
    "room\_temperature": 10,
    "location": {
         "type": "Point",
         "coordinates": [18.17244002, 40.35481002]
    },
    "name": "Room B"
}
{
    "id": "Room3",
    "type": "Room",
    "room\_temperature": 22,
    "location": \{
         "type": "Point",
         "coordinates": [18.17244003, 40.35481003]
    },
    "name": "Room C"
}
```

Listing 3: Gym rooms: Data model definition.

Gym room assets: JSON data model—Listing 3 reports the three data models corresponding to the same rooms of the given gym. The structure of these data models is homogeneous and provides just the specific temperature for each gym room, beyond the corresponding id, type, name, and location attributes. Each room temperature is periodically detected and transmitted to the context broker. In this way, it is possible to identify eventual rooms that present environmental conditions that may be prohibitive compared to the physiological state of a user located nearby.

Based on the data transmitted through the aforementioned data models, the context broker manages to keep the states of the real world actors/assets aligned with their digital counterparts. However, the context broker limits itself to maintaining the current state of the monitored actors/assets, appropriately reporting situations that are expressly declared as relevant for the context broker, but above all for complementary software systems. The mechanism provided by the context broker for this purpose is represented by the subscription.

```
curl -v <IP\_ADDRESS>:<PORT>/v2/subscriptions -s -S --header 'Content-Type:
    application/json' \
    -d @- <<EOF
    {
        "description": "A subscription to get info about User",
        "subject": {
            "entities": [
            {
                "idPattern": ".*",
                "type": "User"
            }
            ],
            "condition": {
                "attrs": [
                    "temperature","heartbeat"
                ],
                "expression": {
                    "q": "temperature>37;heartbeat>68;"
                }
            }
```

```
        },
        "notification": {
            "http": {
                "url": "http://<IP\_address>:<port>/subscription"
            },
            "attrs": [
                "temperature","heartbeat","location"
            ]
        },
        "expires": "2040-01-01T14:00:00.00Z",
        "throttling": 5
    }
```

Listing 4: Context broker subscription: REST call definition.

The REST call to define the case study's subscription is reported in the Listing 4. Relevant attributes in the REST call's payload are subject (rows from 5 to 20) and notification (rows from 21 to 28) attributes. Subject attribute includes two sub-attributes namely entities and condition. Entities sub-attribute is used to identify the specific entities the context broker has to monitor. In this case, all entities of User type have to be monitored. Condition sub-attribute is then used to express the conditions on monitored entities to be checked. In this case, User-type entities' attributes to assess are temperature and heartbeat with respect to 37°C and 68 beats per minute (bpm) threshold values. These values are generally considered as regular for an individual.

Following these settings, the context broker has been instructed to generate an alert when the user's temperature and heartbeat is greater than 37 and 68, respectively.

Instead, the notification sub-attribute is used to express the action to be done when the pre-configured conditions on at least one monitored entity are met. In this case, an HTTP call towards the http://<IP_ADDRESS>:<PORT>/subscription end-point is issued by making available the detected user's temperature, heartbeat rate, and location values. In this case, the indicated end-point represents the edge instance of Telegram BOT sender service, which sends notifications to the target users.

After introducing the involved data models and defining the subscription on the context broker instance, it is possible to test the subscription by increasing the body temperature of the only registered user. For simplicity's sake, we are simulating that the user's temperature has increased as a result of the physical activity performed or currently being performed. Indeed, by leaving the heartbeat value unchanged at 70 for the given user, the corresponding condition is already satisfied for the purposes of activating the subscription (Listing 5).

```
curl <IP\_ADDRESS>:<PORT>/v2/entities/User1/attrs/temperature/value -s -S --
    header 'Content-Type: text/plain' -X PUT -d 37.1
```

Listing 5: Context broker subscription test: REST call for increasing user's temperature and heartbeat rate values.

Following the user's temperature update, the corresponding call to the edge instance of Telegram BOT sender service is performed, generating an Telegram BOT-based alert to the given user. Indeed, the business logic implemented by such a service provides the following: (1) it receives as parameters user's temperature, heartbeat rate, and location, (2) it uses the user's location to individuate gym rooms which are located within a radius of 500 m from the user's GPS position and which have an internal temperature that is less than 11°C for example, (3) by finding at least one room that meets these conditions, it suggests the user to avoid accessing it through a message transmitted by the *App4Health*'s Telegram BOT.

Observing the state of the relevant entities for the case study shown in Table 2, it can be noted that: (1) user's temperature and heartbeat rate values satisfy the subscription's conditions, so the HTTP call to the Telegram BOT sender service is issued, (2) Telegram BOT sender service issues a geo-query on the context broker's database to check the presence of

TABLE 2 Case study: Relevant entities state.

User	Temperature	37.1
	Heartbeat rate	70
	Location	18.17244001, 40.35481001
Room A	Temperature	23
	Location	18.17244001, 40.35481001
Room B	Temperature	10
	Location	18.17244002, 40.35481002
Room C	Temperature	22
	Location	18.17244003, 40.35481003

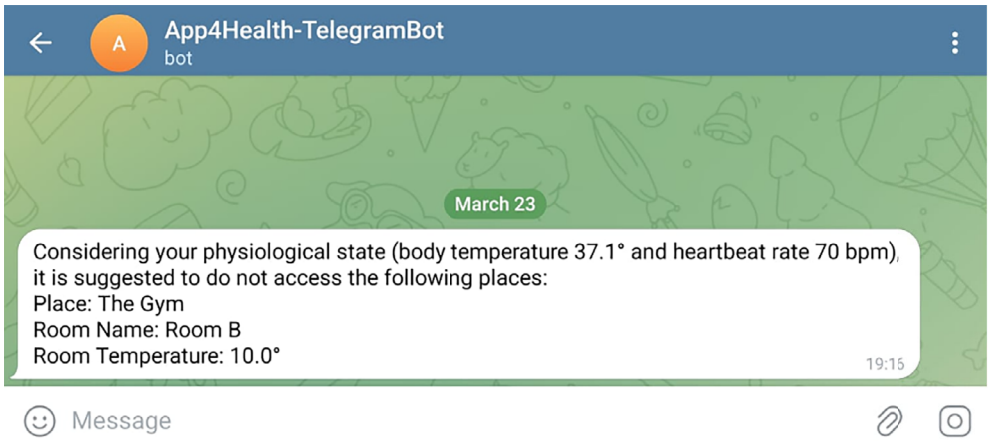

FIGURE 10 Telegram BOT sender service: *App4Health* Telegram BOT.

gym rooms with a temperature lower than 11°C within a radius of 500 m of the user's GPS position. In this case, such a check identifies Room B as a place not to be accessed by the user, considering his body temperature values and heartbeat rate, and then (3) Telegram BOT sender service prepares and delivers a message text to the user's smartphone using the *App4Health* Telegram BOT.

It is important to underline that the threshold values used for the room temperature, the range of action, and the user's temperature and heartbeat rate are not static, but can also be appropriately changed at runtime.

Figure 10 shows the screenshot of the message text received by the registered user from the **App4Health** Telegram BOT. Such a message is prepared by matching a specific generated code among the predefined code-indexed text templates.

5.4 | Final considerations

Considering the results already scientifically documented regarding the semantic aspects of the RS solution we propose,[108] the platform assessment has focused on its architecture. The key factors we have considered for the platform evaluation are related to the scalability of the most solicited services that *App4Health* implements. In particular, these services are the context broker and the Telegram BOT sender components which play a pivotal role to data management and notification delivery, especially in terms of corresponding behavior and performance.

The results we have obtained can be used to give an answer to the RQs that were previously identified in Section 1. For each of these RQs a possible answer will be provided.

5.4.1 | RQ1: How to design a possible architecture of an RS that uses IoT and semantic technologies in order to formulate best practices to suggest to its user?

By observing the results obtained by the architecture evaluation, it is possible to state that the RS platform discussed in section 3 provides details on how to design the software architecture of an RS that can refer to semantics and rule-based approaches to formulate best practices for its user by gathering, processing, monitoring, and analyzing data coming from IoT. Regarding this, the contributions to data management and notification service respectively offered by the context broker and the Telegram BOT sender play a pivotal role, especially in terms of guaranteed performances.

TABLE 3 Lessons learned, recommendations and possible interventions.

Lesson learned	Recommendation/suggested intervention	Exampleof intervention/ implementation
Automation is essential but not sufficient for effective testing in complex projects and large organizations.	People involved in application testing must be able to interpret and understand the testing activities performed to identify complex scenarios that automated tests might miss or overlook (e.g., those involving real-time health data) and that would need manual testing.	Implementing protocols where testers manually review and assess specific cases identified by automated systems (e.g., unusual sensor readings, unexpected user behavior patterns).
Domain-specific approaches can significantly enhance the effectiveness of testing tools, especially in specific domains.	The definition of a test strategy that considers contextual conditions is essential for the successful integration of any testing tools or modules.	Creating test scripts that simulate specific and diverse health-related scenarios (e.g., different stress levels, physical activities, environmental conditions) and assessing how the system behaves accordingly.
Producing flexible outputs that can integrate with organizational processes is crucial for tool adoption.	Defining a test strategy that can be customized to align with reference standards and regulatory requirements is advisable for the successful integration of any testing tools or modules.	Formatting test reports to include compatibility and interoperability with health-related standards (e.g., health-compliance metrics, privacy checks on patient data).
Low quality levels in data quality characteristics indicate underlying issues that could require effort to be identified and addressed.	Defining specific business rules, such as syntax requirements and data value ranges, can help improve data quality.	Implementing real-time data validation solutions capable of checking for anomalies in data, especially those gathered from wearable devices.
Establishing evaluation requirements for clear data quality is crucial for successful initiatives, but users may experience various challenges related to business rules, especially in the early period when the same rules are introduced.	Adopting incremental training sessions and user feedback loops can help users adapt more quickly to data quality standards and business rules.	Introducing new data quality control protocols in stages (e.g., involving pilot groups first and using their feedback to refine protocols before wider implementation).
Test reports play a vital role in system testing assessment and can be useful for additional testing activities.	Effective and comprehensive test reports can streamline the evaluation process, offer actionable insights, and enable the scheduling of new focused test activities with reduced effort.	Enhancing test reports by highlighting trends in sensor data accuracy, system response time, and user compliance to optimize system performance and user engagement.
The stability of results across repeated experiments is crucial for achieving meaningful and reproducible outcomes.	Although limitations in experiment repetitions may exist, it is important to ensure consistent testing conditions and replicate experiments to verify the stability and reliability of health-related recommendations.	Standardizing the test environment and procedures (e.g., using predefined sensor configurations and user profiles) and repeating tests to confirm the consistency of the health recommendations generated by the system.

5.4.2 | RQ2: How to collect, process, and analyze data from MCS scenarios as well as personal sensing contexts to make it possible for users to monitor and preserve their health?

Providing a context broker solution as a supportive software component to data management for the *App4Health* platform, it becomes possible to efficiently and effectively collect and preprocess data coming from MCS scenarios and corresponding proximity environments. These data are then opportunely processed and analyzed using both a semantic-based and a rule-base approach to formulate and propose best practices to users for monitoring and preserving their health status. Following this procedure, the *App4Health* platform is capable of offering functionalities useful to support users in monitoring their health status. In essence, *App4Health* platform is capable of providing recommendations at cloud and edge/fog tiers by following a semantic-based and a rule-based approach, respectively. This RQ is specifically addressed in Sections 4.1.1 and 4.2.1.

5.4.3 | RQ3: How to support the formulation of best practices for users by reducing both the internet and cloud dependencies in order to provide recommendations/suggestions according to contextual conditions and in near-/real-time?

Being compliant with the ECC paradigm, the *App4Health* MSA enables the edge/fog deployment of microservices capable of offering proximity-value-added services to users. The ECC paradigm aims to reduce both the internet and cloud dependencies of the services implemented by the *App4Health* platform. Indeed, in case of rule-based recommendation formulation, the proposed platform achieves the ambitious goal to provide proximity-added-value services in near-real-time and at the edge of the internet. Indeed, among the possible offered services, it is possible to include the formulation of best practices for users. This RQ is tackled in Section 3.

5.5 | Lessons learned

This study has yielded valuable insights and recommendations that can be leveraged to support the testing of similar future software systems and their practical applications in real-world scenarios. A collection of lessons learned, along with the corresponding recommendations we derived (as well as a series of implementation examples that we propose), is reported in Table 3.

6 | CONCLUSIONS AND FUTURE WORK

The present paper focuses on the architecture design of a RS in the context of MCS, healthcare and wellness. The RS leverages an ad-hoc developed cloud-based platform, named *App4Health*. For deriving the suggestions to provide to users during their training activities, the *App4Health* platform refers to both a semantic- and rule-based approach. For this reason, the proposed platform architecture leverages a modular ontology and a context broker instance, respectively. These two different solutions make it possible to formulate recommendations to users at the cloud and edge/fog tiers. In this last case, recommendations can be considered as proximity-added-value services to be provided at near real-time. The *App4Health*'s multi-layer architecture implements the ECC paradigm and meets the interoperability requirements of complex apps relying on heterogeneous sensors. For evaluating the proposed architecture, we have performed a specific platform assessment through a comparative performance evaluation. The software components of the proposed architecture we have considered for the system assessment are the context broker and the Telegram BOT sender. Indeed, we believe that the major criticisms for the platform performance can affect the hardware and software components located within the edge/fog tier. For the purpose of evaluating the proposed architecture, two testbed configurations were developed to carry out a comparative stress test of the context broker and the Telegram BOT sender components deployed at the edge/fog tier. The comparison carried out aims to identify the performance degradation threshold of these components, assuming that they can be deployed on edge-level hardware devices with different technical specifications. For the testbed development, the smart edge devices that we opted for deploying the edge/fog software components were respectively a Raspberry Pi 3-Model B+ and a Raspberry PI 4 Cluster device. Moreover, for the comparative stress test, we have developed a Python module for simulating thread-based REST calls

addressed to the Fiware Orion context broker and for forcing the Telegram BOT sender to perform a massive message sending to the preconfigured Telegram BOT. By analyzing the comparative performance results, it is possible to note that the response times obtained by deploying the examined *App4Health* services on two different hardware devices and referring to different software solutions follow a linear trend as processing and storage capacity increases, at least up to a certain workload level. For better understanding how the proposed RS works, a case study is further discussed.

Regarding future work, it is first planned to adopt NGSI-LD compliant data models[148] and explore an alternative approach to internal building GPS tracking, specifically matching user positions to the actual gym room coordinates. Moreover, a more complete and exhaustive overall system assessment will be conducted by referring to specific simulators, such as PureEdgeSim[149] and EdgeCloudSim.[150] In this way, it will be also possible to monitor the overall resource usage and network condition variations in order to provide a richer and more comprehensive analysis. In this case, we have indeed opted to implement a custom stress test instead of one of these simulators because our first intention was to assess the behavior of the most targeted software component of the architecture we propose. In this regard, we believe that the results obtained are nevertheless relevant, being able to provide certainties regarding an acceptable degree of stability, scalability and performance of the most stressed components of the proposed RS, although not referable to the entire architecture. In addition, the *App4Health* platform architecture will be further enhanced by adding specific measures to adequately guarantee aspects related to security, data trustworthiness, and privacy. These aspects are planned to be adequately addressed in a forthcoming version of the *App4Health* platform, considering their significance corresponding to the data the same platform must gather, manage and share. Finally, another significant shortcoming of the platform we propose is related to stringent limitations in terms of inter-device trustworthiness and data privacy. Regarding the trustworthiness, it is worth anticipating that an evolutionary step of the *App4Health* platform is already under study, with the aim of documenting it in a forthcoming research paper. Once that aspect will be dealt with, we plan to address data privacy challenges as well.

ACKNOWLEDGMENTS

This work is partially supported by ICSC-Italian center on Supercomputing and Italian Research Center on High Performance Computing, Big Data and Quantum Computing, funded by European Union - NextGenerationEU, and by CETMA DIHSME. Open access publishing facilitated by Universita del Salento, as part of the Wiley - CRUI-CARE agreement.

FUNDING INFORMATION

None.

CONFLICT OF INTEREST STATEMENT

The authors declare no potential conflict of interests.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

ORCID

Angelo Martella https://orcid.org/0000-0002-1082-7293
Antonella Longo https://orcid.org/0000-0002-6902-0160
Antonio Esposito https://orcid.org/0000-0002-2004-4815

REFERENCES

1. Roy D, Dutta M. A systematic review and research perspective on recommender systems. *J Big Data*. 2022;9(1):1-36.
2. Lu J, Wu D, Mao M, Wang W, Zhang G. Recommender system application developments: a survey. *Decis Support Syst*. 2015;74:12-32. doi:10.1016/j.dss.2015.03.008
3. Shah K, Salunke A, Dongare S, Antala K. Recommender systems: an overview of different approaches to recommendations. *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. IEEE; 2017:1-4.
4. Binucci C, De Luca F, Di Giacomo E, Liotta G, Montecchiani F. Designing the content analyzer of a travel recommender system. *Expert Syst Appl*. 2017;87:199-208.

5. Sun Z, Han L, Huang W, et al. Recommender systems based on social networks. *J Syst Softw*. 2015;99:109-119.
6. Gomez-uribe C, Hunt N. The Netflix recommender System_Algorithms; 2015.
7. Poongodi T, Krishnamurthi R, Indrakumari R, Suresh P, Balusamy B. *Wearable Devices and IoT*. Springer International Publishing; 2019:245-273.
8. Takiddeen N, Zualkernan I. Smartwatches as IoT edge devices: a framework and survey. *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE; 2019:216-222.
9. Assaba C, Gite S. *IoT and Smartphone-Based Remote Health Monitoring Systems*. Springer International Publishing; 2021:243-253.
10. Giorgi G, Galli A, Narduzzi C. Smartphone-based IOT systems for personal health monitoring. *IEEE Instrum Meas Mag*. 2020;23(4):41-47. doi:10.1109/mim.2020.9126070
11. Mishra SS, Rasool A. IoT health care monitoring and tracking: a survey. *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE; 2019:1052-1057.
12. John Dian F, Vahidnia R, Rahmati A. Wearables and the Internet of Things (IoT), applications, opportunities, and challenges: a survey. *IEEE Access*. 2020;8:69200-69211. doi:10.1109/access.2020.2986329
13. Yang Y, Wang H, Jiang R, Guo X, Cheng J, Chen Y. A review of IoT-enabled mobile healthcare: technologies, challenges, and future trends. *IEEE Internet Things J*. 2022;9(12):9478-9502. doi:10.1109/jiot.2022.3144400
14. Friedewald M, Raabe O. Ubiquitous computing: an overview of technology impacts. *Telemat Inform*. 2011;28(2):55-65. doi:10.1016/j.tele.2010.09.001
15. Bangare PS, Patil KP. Security issues and challenges in Internet of Things (IOT) system. *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*. IEEE; 2022:91-94.
16. Kurda R, Haje UA, Abdulla MH, Khalid ZM. A review on security and privacy issues in IoT devices; 2022.
17. Gupta SK, Chandan RR, Shukla R, Singh P, Pandey AK, Jaiswal AK. Heterogeneity issues in IoT-driven devices and services. *J Auton Intell*. 2023;6(2):588. doi:10.32629/jai.v6i2.588
18. Fu W, Peng Z, Wang S, Xu Y, Li J. Deeply fusing reviews and contents for cold start users in cross-domain recommendation systems. *Proc AAAI Conf Artif Intell*. 2019;33:94-101.
19. Gao C, Zheng Y, Li N, et al. A survey of graph neural networks for recommender systems: challenges, methods, and directions. *ACM Trans Recomm Syst*. 2023;1(1):1-51.
20. Zhao X, Wang M, Zhao X, et al. Embedding in recommender systems: a survey. arXiv preprint arXiv:2310.18608, 2023.
21. Adomavicius G, Zhang J. Impact of data characteristics on recommender systems performance. *ACM Trans Manag Inf Syst*. 2012;3(1):1-17.
22. Heinrich B, Hopf M, Lohninger D, Schiller A, Szubartowicz M. Data quality in recommender systems: the impact of completeness of item content data on prediction accuracy of recommender systems. *Electron Mark*. 2019;31(2):389-409. doi:10.1007/s12525-019-00366-7
23. Schafer JB. *The Application of Data-Mining to Recommender Systems*. IGI Global; 2009:45-50.
24. Rodrigues MB, Silva DGOM, Durão FA. User models development based on cross-domain for recommender systems. *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web Webmedia*. ACM; 2016.
25. Zhang Q, Wu D, Lu J, Liu F, Zhang G. A cross-domain recommender system with consistent information transfer. *Decis Support Syst*. 2017;104:49-63. doi:10.1016/j.dss.2017.10.002
26. Cantador I, Cremonesi P. Tutorial on cross-domain recommender systems. *Proceedings of the 8th ACM Conference on Recommender Systems*. ACM; 2014.
27. Cremonesi P, Tripodi A, Turrin R. Cross-domain recommender systems. *2011 IEEE 11th International Conference on Data Mining Workshops*. IEEE; 2011:496-503.
28. Chen J, Dong H, Wang X, Feng F, Wang M, He X. Bias and debias in recommender system: a survey and future directions. *ACM Trans Inf Syst*. 2023;41(3):1-39.
29. Rath CK, Mandal AK, Sarkar A. Microservice based scalable IoT architecture for device interoperability. *Comput Stand Inter*. 2023;84:103697.
30. Antonios P, Konstantinos K, Christos G. A systematic review on semantic interoperability in the IoE-enabled smart cities. *Internet Things*. 2023;22:100754.
31. Walker DM, Tarver WL, Jonnalagadda P, Ranbom L, Ford EW, Rahurkar S. Perspectives on challenges and opportunities for interoperability: findings from key informant interviews with stakeholders in Ohio. *JMIR Med Inform*. 2023;11(1):e43848.
32. Longo A, Zappatore M, De Matteis A. An osmotic computing infrastructure for urban pollution monitoring. *Softw Pract Exp*. 2020;50(5):533-557. doi:10.1002/spe.2721
33. Ganti R. Mobile crowdsensing: current state and future challenges. *IEEE Commun Mag*. 2011;49:32-39.
34. Amazon Mechanical Turk. Accessed February 20, 2024. https://www.mturk.com
35. Musthag M, Ganesan D. Labor dynamics in a mobile micro-task market. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM; 2013:641-650.
36. Uber. Accessed February 20, 2024. https://www.uber.com/
37. Gigwalk. We've got your brand's back—Gigwalk. Accessed February 20, 2024. https://www.gigwalk.com/
38. Taskrabbit. Same day handyman, moving & mounting services. Accessed February 20, 2024. https://www.taskrabbit.com/
39. grubhub.com. Prepare your taste buds … . Accessed February 20, 2024. https://www.grubhub.com/
40. Instacart. Accessed February 20, 2024. https://www.instacart.com/
41. Waze. Driving directions, live traffic & road conditions updates. Accessed February 20, 2024. https://www.waze.com/

42. openstreetmap.org. OpenStreetMap. Accessed February 20, 2024. https://www.openstreetmap.org/
43. Chen L, Shahabi C. Spatial crowdsourcing: challenges and opportunities. *IEEE Data Eng Bull*. 2016;39(4):14-25.
44. Capponi A, Fiandrino C, Kantarci B, Foschini L, Kliazovich D, Bouvry P. A survey on mobile crowdsensing systems: challenges, solutions, and opportunities. *IEEE Commun Surv Tutor*. 2019;21(3):2419-2465.
45. Gao C, Kong F, Tan J. HealthAware: tackling obesity with health aware smart phone systems. *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE; 2009:1549-1554.
46. Chen G, Yan B, Shin M, Kotz D, Berke E. MPCS: mobile-phone based patient compliance system for chronic illness care. *2009 6th Annual International Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous*. IEEE; 2009:1-7.
47. Reddy S, Parker A, Hyman J, Burke J, Estrin D, Hansen M. Image browsing, processing, and clustering for participatory sensing: lessons from a dietsense prototype. *Proceedings of the 4th Workshop on Embedded Networked Sensors*. ACM; 2007:13-17.
48. Mohan P, Padmanabhan VN, Ramjee R. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. ACM; 2008:323-336.
49. Hasenfratz D, Saukh O, Sturzenegger S, Thiele L. Participatory air pollution monitoring using smartphones. *Mob Sens*. 2012;1:1-5.
50. Sivaraman V, Carrapetta J, Hu K, Luxan BG. HazeWatch: a participatory sensor system for monitoring air pollution in Sydney. *38th Annual IEEE Conference on Local Computer Networks-Workshops*. IEEE; 2013:56-64.
51. Liu L, Liu W, Zheng Y, Ma H, Zhang C. Third-eye: a mobilephone-enabled crowdsensing system for air quality monitoring. *Proc ACM Interact Mob Wearable Ubiquitous Technol*. 2018;2(1):1-26.
52. Reddy S, Samanta V. *Urban Sensing: Garbage Watch*. UCLA Center for Embedded Networked Sensing; 2011.
53. Bonino D, Alizo MTD, Pastrone C, Spirito M. WasteApp: smarter waste recycling for smart citizens. *2016 International Multidisciplinary Conference on Computer and Energy Science (SpliTech)*. IEEE; 2016:1-6.
54. Talasila M, Curtmola R, Borcea C. Mobile crowd sensing; 2015.
55. Crowd YY. Monitoring using mobile phones. *2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics*. IEEE; 2014.
56. Coluccia A, Fascista A. A review of advanced localization techniques for crowdsensing wireless sensor networks. *Sensors*. 2019;19(5):988. doi:10.3390/s19050988
57. Namiot D, Schneps-Schneppe M. On crowd sensing back-end. *International Conference on Data Analytics and Management in Data Intensive Domains*. CEUR-WS.org; 2016.
58. Merlino G, Arkoulis S, Distefano S, Papagianni C, Puliafito A, Papavassiliou S. Mobile crowdsensing as a service: a platform for applications on top of sensing clouds. *Future Gener Comput Syst*. 2016;56:623-639. doi:10.1016/j.future.2015.09.017
59. Dasari VS, Kantarci B, Pouryazdan M, Foschini L, Girolami M. Game theory in mobile crowdsensing: a comprehensive survey. *Sensors*. 2020;20(7):2055. doi:10.3390/s20072055
60. Tezza R, Vargas GL. Scale to measure usability in mobile crowd sensing (MCS) applications. *J Stat Manag Syst*. 2021;25(3):501-520. doi:10.1080/09720510.2021.1878631
61. He D, Chan S, Guizani M. User privacy and data trustworthiness in mobile crowd sensing. *IEEE Wirel Commun*. 2015;22(1):28-34. doi:10.1109/mwc.2015.7054716
62. Liu J, Shen H, Narman HS, Chung W, Lin Z. A survey of mobile crowdsensing techniques: a critical component for the Internet of Things. *ACM Trans Cyber-Phys Syst*. 2018;2(3):1-26.
63. Heigl F, Kieslinger B, Paul KT, Uhlik J, Dörler D. Toward an international definition of citizen science. *Proc Natl Acad Sci*. 2019;116(17):8089-8092. doi:10.1073/pnas.1903393116
64. Rutten M, Minkman E, Sanden M. How to get and keep citizens involved in mobile crowd sensing for water management? A review of key success factors and motivational aspects. *WIREs Water*. 2017;4(4):e1218. doi:10.1002/wat2.1218
65. Zhu C, Zhou H, Leung VCM, Wang K, Zhang Y, Yang LT. Toward big data in green city. *IEEE Commun Mag*. 2017;55(11):14-18. doi:10.1109/mcom.2017.1700142
66. Aly H, Basalamah A, Youssef M. Automatic rich map semantics identification through smartphone-based crowd-sensing. *IEEE Trans Mob Comput*. 2017;16(10):2712-2725. doi:10.1109/tmc.2016.2645150
67. Liu Z, Jiang S, Zhou P, Li M. A participatory urban traffic monitoring system: the power of bus riders. *IEEE Trans Intell Transp Syst*. 2017;18(10):2851-2864. doi:10.1109/tits.2017.2650215
68. Wang Z, Guo B, Yu Z, et al. PublicSense: a crowd sensing platform for public facility management in smart cities. *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. IEEE; 2016:114-120.
69. Neha B, Panda SK, Sahu PK, Sahoo KS, Gandomi AH. A systematic review on osmotic computing. *ACM Trans Internet Things*. 2022;3(2):9. doi:10.1145/3488247
70. Hajjaji Y, Boulila W, Riadh Farah I, Romdhani I, Hussain A. Big data and IoT-based applications in smart environments: a systematic review. *Comput Sci Rev*. 2021;39:100318. doi:10.1016/j.cosrev.2020.100318
71. Mishra S, Jain S. Ontologies as a semantic model in IoT. *Int J Comput Appl*. 2018;42(3):233-243. doi:10.1080/1206212x.2018.1504461
72. European Commission. European cloud-edge technology investment roadmap. 2021. Accessed January 15, 2023. https://ec.europa.eu/newsroom/repository/document/2021-18/European_CloudEdge_Technology_Investment_Roadmap_for_publication_pMdz85DSw6nqPppq8hE9S9RbB8_76223.pdf
73. Jannach D, Zanker M, Felfernig A, Friedrich G. *Recommender Systems: An Introduction*. Cambridge University Press; 2010.

74. Crespo RG, Martínez OS, Lovelle JMC, García-Bustelo BCP, Gayo JEL, Pablos PO. Recommendation system based on user interaction data applied to intelligent electronic books. *Comput Human Behav*. 2011;27(4):1445-1449.
75. Bobadilla J, Serradilla F, Bernal J. A new collaborative filtering metric that improves the behavior of recommender systems. *Knowl Based Syst*. 2010;23(6):520-528.
76. Yoshii K, Goto M, Komatani K, Ogata T, Okuno HG. An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model. *IEEE Trans Audio Speech Lang Process*. 2008;16(2):435-447.
77. Wang SL, Wu CY. Application of context-aware and personalized recommendation to implement an adaptive ubiquitous learning system. *Expert Syst Appl*. 2011;38:10831-10838.
78. Salehi M, Kmalabadi IN. A hybrid attribute-based recommender system for E-learning material recommendation. *IERI Procedia*. 2012;2:565-570.
79. Bobadilla J, Serradilla F, Hernando A. Collaborative filtering adapted to recommender systems of e-learning. *Knowl Based Syst*. 2009;22(4):261-265.
80. García-Crespo Á, López-Cuadrado JL, Colomo-Palacios R, González-Carrasco I, Ruiz-Mezcua B. Sem-fit: a semantic based expert system to provide recommendations in the tourism domain. *Expert Syst Appl*. 2011;38(10):13310-13319.
81. Rahul DH, Singh D. A review of trends and techniques in recommender systems. *2019 4th International Conference on Internet of Things: Smart Innovation and Usages (IoT-SIU)*. IEEE; 2019:1-8.
82. Mohamed MH, Khafagy MH, Ibrahim MH. Recommender systems challenges and solutions survey. *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*. IEEE; 2019:149-155.
83. Zhu F, Wang Y, Chen C, Zhou J, Li L, Liu G. Cross-domain recommendation: challenges, progress, and prospects. arXiv preprint arXiv:2103.01696, 2021.
84. Middleton SE, Roure DD, Shadbolt NR. Ontology-based recommender systems. *Handbook on Ontologies*. Springer; 2004:477-498.
85. Bridge D, Göker MH, McGinty L, Smyth B. Case-based recommender systems. *Knowl Eng Rev*. 2005;20(3):315-320.
86. Felfernig A, Burke R. Constraint-based recommender systems: technologies and research issues. *Proceedings of the 10th International Conference on Electronic Commerce*. ACM; 2008:1-10.
87. Guo Q, Zhuang F, Qin C, et al. A survey on knowledge graph-based recommender systems. *IEEE Trans Knowl Data Eng*. 2020;34(8):3549-3568.
88. Tarus JK, Niu Z, Mustafa G. Knowledge-based recommendation: a review of ontology-based recommender systems for e-learning. *Artif Intell Rev*. 2018;50:21-48.
89. Dong H, Hussain FK, Chang E. A service concept recommendation system for enhancing the dependability of semantic service matchmakers in the service ecosystem environment. *J Netw Comput Appl*. 2011;34(2):619-631.
90. Mohanraj V, Chandrasekaran M, Senthilkumar J, Arumugam S, Suresh Y. Ontology driven bee's foraging approach based self adaptive online recommendation system. *J Syst Softw*. 2012;85(11):2439-2450.
91. Champiri ZD, Shahamiri SR, Salim SSB. A systematic review of scholar context-aware recommender systems. *Expert Syst with Appl* 2015; 42(3):1743-1758.
92. Abu-Issa A, Nawawreh H, Shreteh L, et al. A smart city mobile application for multitype, proactive, and context-aware recommender system. *2017 International Conference on Engineering and Technology (ICET)*. IEEE; 2017:1-5.
93. Panniello U, Tuzhilin A, Gorgoglione M. Comparing context-aware recommender systems in terms of accuracy and diversity. *User Model User-Adapt Interact*. 2014;24(1):35-65.
94. Ojagh S, Malek MR, Saeedi S, Liang S. An Internet of Things (IoT) approach for automatic context detection. *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE; 2018:223-226.
95. Adomavicius G, Tuzhilin A. Context-aware recommender systems. *Recommender Systems Handbook*. Springer; 2011:217-253.
96. Yong B, Xu Z, Wang X, et al. IoT-based intelligent fitness system. *J Parallel Distrib Comput*. 2018;118:14-21.
97. Nag N, Pandey V, Jain RC. Endogenous and exogenous multi-modal layers in context aware recommendation systems for health. arXiv preprint arXiv:1808.06468, 2018.
98. Casino F, Patsakis C, Batista E, Postolache O, Martínez-Ballesté A, Solanas A. Smart healthcare in the IoT era: a context-aware recommendation example. *2018 International Symposium in Sensing and Instrumentation in IoT Era (ISSI)*. IEEE; 2018:1-4.
99. Aipe A, Sundararaman MN, Ekbal A. Sentiment-aware recommendation system for healthcare using social media. arXiv preprint arXiv:1909.08686, 2019.
100. Almeida JR, Monteiro E, Silva LB, Sierra AP, Oliveira JL. A recommender system to help discovering cohorts in rare diseases. *2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE; 2020:25-28.
101. Jie M, HuiMing Y, Yizhuo C. Research on ordering recommendation system of traditional Chinese medical health preserving ontology model based on context-aware environment. *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*. IEEE; 2020:629-632.
102. Mojarad R, Attal F, Chibani A, Amirat Y. Context-aware adaptive recommendation system for personal well-being services. *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE; 2020:192-199.
103. Fiware Orion context broker. 2024. Accessed March 28, 2024. https://fiware-orion.readthedocs.io/en/master/
104. Fiware. 2024. Accessed March 28, 2024. https://www.fiware.org/
105. Fiware cygnus connector. 2024. Accessed March 28, 2024. https://fiware-cygnus.readthedocs.io/en/latest/
106. Fiware generic enablers catalogue. 2024. Accessed March 28, 2024. https://www.fiware.org/catalogue/
107. i4trust: collaboration program targeted to accelerate the creation of data spaces. 2024. Accessed March 28, 2024. https://i4trust.org/

108. Zappatore M, Longo A, Martella A, Martino BD, Esposito A, Gracco SA. Semantic models for IoT sensing to infer environment–wellness relationships. *Future Gener Comput Syst*. 2023;140:1-17. doi:10.1016/j.future.2022.10.005
109. Villari M, Fazio M, Dustdar S, Rana O, Ranjan R. Osmotic computing: a new paradigm for edge/cloud integration. *IEEE Cloud Comput*. 2016;3(6):76-83. doi:10.1109/mcc.2016.124
110. Souza A, Cacho N, Noor A, Jayaraman PP, Romanovsky A, Ranjan R. Osmotic monitoring of microservices between the edge and cloud. *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE; 2018:758-765.
111. Carnevale L, Celesti A, Galletta A, Dustdar S, Villari M. From the cloud to edge and IoT: a smart orchestration architecture for enabling osmotic computing. *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*. IEEE; 2018:419-424.
112. Fazio M, Celesti A, Ranjan R, Liu C, Chen L, Villari M. Open issues in scheduling microservices in the cloud. *IEEE Cloud Comput*. 2016;3(5):81-88. doi:10.1109/MCC.2016.112
113. AirCare: take control of the air you breathe. 2024. Accessed February 20, 2024. https://www.aircare.it/en/home-eng/
114. Fiware open API specifications. 2022. Accessed January 15, 2023. https://github.com/fiware/specifications
115. Hai R, Quix C, Jarke M. Data lake concept and systems: a survey. arXiv preprint arXiv:2106.09592, 2021.
116. Walker C, Alrehamy H. Personal data lake with data gravity pull. *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*. IEEE; 2015:160-167.
117. Stein B, Morrison A. The enterprise data lake: better integration and deeper analytics. *PwC Technol Forecast Rethink Integr*. 2014;1(1–9):18.
118. Terrizzano IG, Schwarz PM, Roth M, Colino JE. Data wrangling: the challenging journey from the wild to the lake. *7th Biennial Conference on Innovative Data Systems Research*. Asilomar; 2015.
119. Quix C, Hai R, Vatov I. GEMMS: a generic and extensible metadata management system for data lakes. CAiSE Forum. 2016:129.
120. Miloslavskaya N, Tolstoy A. Big data, fast data and data lake concepts. *Procedia Comput Sci*. 2016;88:300-305.
121. International Data Spaces. 2022. Accessed January 15, 2023. https://www.internationaldataspaces.org/
122. Project GAIA-X: a federated data infrastructure as the cradle of a vibrant European ecosystem; executive summary. 2022. Accessed January 15, 2023. https://www.data-infrastructure.eu/GAIAX/Redaktion/EN/Publications/das-projekt-gaia-x-executive-summary.html
123. Data Spaces for effective and trusted data sharing. Accessed February 25, 2024. https://i4trust.org/
124. Teiid: cloud-native data virtualization. 2024. Accessed February 20, 2024. https://teiid.io/
125. mongodb.com. MongoDB: the developer data platform. Accessed February 25, 2024. https://www.mongodb.com/
126. PostgreSQL Global Development Group. PostgreSQL—postgresql.org. Accessed February 25, 2024. https://www.postgresql.org/
127. Apache Software Foundation. Spark. 2023. Accessed January 15, 2023. https://spark.apache.org/
128. Martino BD, Esposito A, Cretella G. Semantic representation of cloud patterns and services with automated reasoning to support cloud application portability. *IEEE Trans Cloud Comput*. 2017;5(4):765-779. doi:10.1109/TCC.2015.2433259
129. Martin D, Burstein M, Hobbs J, et al. OWL-S: semantic markup for web services. W3C Member Submission; 2004;22:4.
130. Gasmi A, Tamani N, Faucher C, Ghamri-Doudane Y. OAISIS: an ontological-based approach for interlinking CrowdSensing information systems. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE; 2016:003995-004000.
131. Time ontology in OWL—W3C candidate recommendation 26 March 2020. 2022. Accessed August 30, 2023. https://www.w3.org/TR/owl-time/
132. Basic Geo (WGS84 lat/long) Vocabulary. 2022. Accessed August 30, 2023. https://www.w3.org/2003/01/geo/
133. CoDAMoS—context-driven adaptation of mobile services ontology. 2022. Accessed August 30, 2023. https://distrinet.cs.kuleuven.be/projects/CoDAMoS/ontology/
134. Keil JM, Schindler S. Comparison and evaluation of ontologies for units of measurement. *Semant Web*. 2018;10(1):33-51. doi:10.3233/sw-180310
135. QUDT catalog—quantities, units, dimensions and data types ontologies. 2022. Accessed August 30, 2023. https://www.qudt.org/2.1/catalog/qudt-catalog.html
136. FAIRsharing Team. FAIRsharing record for: quantities, units, dimensions and types; 2015.
137. Ontology of Units of Measure. 2022. Accessed August 30, 2023. https://bioportal.bioontology.org/ontologies/OM
138. Rijgersberg H, Assem vM, Top J. Ontology of units of measure and related concepts. *Semant Web*. 2013;4(1):3-13. doi:10.3233/sw-2012-0069
139. Units of Measurement Ontology. 2022. Accessed August 30, 2023. https://bioportal.bioontology.org/ontologies/UO
140. Gkoutos GV, Schofield PN, Hoehndorf R. The units ontology: a tool for integrating units of measurement in science. *Database*. 2012;2012:bas033. doi:10.1093/database/bas033
141. Semantic sensor network ontology—W3C recommendation 19 October 2017. 2022. Accessed August 30, 2023. https://www.w3.org/TR/vocab-ssn
142. Wang J, Helal S, Wang Y, Zhang D. WSelector: a multi-scenario and multi-view worker selection framework for crowd-sensing. *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and its Associated Workshops (UIC-ATC-ScalCom)*. IEEE; 2015:54-61.
143. Zappatore M, Longo A, Bochicchio MA, Zappatore D, Morrone AA, De Mitri G. Mobile crowd sensing-based noise monitoring as a way to improve learning quality on acoustics. *2015 International Conference on Interactive Mobile Communication Technologies and Learning (IMCL)*. IEEE; 2015:96-100.

144. Longo A, Zappatore M, Bochicchio MA. Collaborative learning from mobile crowd sensing: a case study in electromagnetic monitoring. *2015 IEEE Global Engineering Education Conference (EDUCON)*. IEEE; 2015:742-750.
145. Zappatore M, Loglisci C, Longo A, Bochicchio MA, Vaira L, Malerba D. Trustworthiness of context-aware urban pollution data in mobile crowd sensing. *IEEE Access*. 2019;7:154141-154156. doi:10.1109/ACCESS.2019.2948757
146. Raspberry Pi 3-Model B+. 2022. Accessed January 15, 2023. https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/
147. Arduino. 2022. Accessed January 15, 2023. https://www.arduino.cc
148. smartdatamodels.org. Smart data models. Accessed February 26, 2024. https://smartdatamodels.org/
149. Mechalikh C, Taktak H, Moussa F. PureEdgeSim: a simulation toolkit for performance evaluation of cloud, fog, and pure edge computing environments. *2019 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE; 2019:700-707.
150. Sonmez C, Ozgovde A, Ersoy C. EdgeCloudSim: an environment for performance evaluation of edge computing systems. *Trans Emerg Telecommun Technol*. 2018;29(11):e3493.

How to cite this article: Martella A, Longo A, Zappatore M, Martino BD, Esposito A. A semantically enabled architecture for interoperable edge-cloud continuum applied to the e-health scenario. *Softw: Pract Exper*. 2025;55(3):409-447. doi: 10.1002/spe.3375