STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **16**, August 2026, pp 1760–1789.
Published online in International Academic Press (www.IAPress.org)

Medical Data and Services in Efficient Personal Care Delivery: A Novel Dynamic Semantic Microservice CQRS Framework

Nassim Boukezoula [1], Abdelhamid Malki [1], Mimoun Malki [1], Adel Alti[2*]

[1] *LabRi Laboratory, Ecole Superieure en Informatique, Sidi Bel Abbes, 22000, Algeria*
[2] *LRSD Laboratory, University Ferhat Abbas Sétif-1, Sétif, Algeria*

Abstract Health crises often overwhelm people, causing significant stress for physicians not only during the active operational period but also throughout the entire patient care life cycle.In these kinds of scenarios, the pressure on medical professionals can lead to service inconsistencies and increased complexity in delivering ongoing and coordinated healthcare services. Previous research studies have been mostly focused on wireless biomedical sensors and big data analytics, often neglecting patient context under several moving locations where service tends to break as a result of unexpected events while running. To address this gap, we explore Command Query Responsibility Segregation (CQRS) pattern and ConteXt-aware Agent-Oriented Semantic MicroServices (CxASMS), to ensure service continuity with less computational complexity. As the case study, we employed pregnant women, which is often used in the health field. Results of our study prove that context-aware semantic models based on CQRS can deal with service volatility induced by context changes while improving accuracy throughout the semantic service life cycle. The proposed platform reuses previous responses and views within the patient and service contexts to optimize the response time. This is the first dynamic semantic CQRS framework in the field of smart health for service continuity using context-aware sensing, which is then properly evaluated using a real-world study.

Keywords CxS-CQRS; Service Continuity; Context-awareness; Matching; Health; CxASMS

DOI: 10.19139/soic-2310-5070-3596

1. Introduction

IoT platforms are increasingly used across domains like smart health [1], logistics [2], farming [3], and Industry 4.0 [4] due to their efficiency, agility, and real-time data access. In healthcare, especially during crises like COVID-19 [5], the massive data generated poses challenges in delivering timely and accurate services. This demands advanced systems capable of integrating heterogeneous health data.
Microservice-based architectures offer modularity and scalability [6], but synchronizing data across services. Each service with its own database is complex. Command Query Responsibility Segregation (CQRS) addresses this by separating update (command) and read (query) operations [7], with event-driven communication enhancing responsiveness. The CQRS pattern is utilized to asynchronously communicate changes and updates between different parts of the system However, CQRS lacks semantic understanding, limiting its ability to manage dynamic, context-rich health environments such as mobile patients, sensor failures, or fluctuating health status where rapid service discovery and adaptation are vital.
To overcome this, we propose a context-aware semantic CQRS microservices-based strategy is required which provides data in a unified format and enhances semantic service discovery and matching. This approach is providing facility to update unavailable service by another equivalent to ensure dynamic data flow transmission and real-time analysis. We aim to explore the potential of context awareness in selection and orchestration of event-driven

*Correspondence to: Nassim Boukezoula (Email: n.boukezoula@esi-sba.dz). LabRi Laboratory, Ecole Superieure en Informatique, Sidi Bel Abbes, 22000, Algeria.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

services to support better decision-making using available microservices. Specifically, we focus on investigating the use of, context-aware microservices with CQRS pattern and intelligent semantic CQRS broker to process users' requests and properly responded to different types of requests (commands, queries). Therefore, our research aims to build an effective semantic CQRS approach to enhance the accuracy of results and optimize treatment time in the context of dynamic distributed smart health services. This research work makes the following contributions :

- We combine the CQRS architectural design pattern with ConteXt-aware Agent-Oriented Semantic MicroServices (CxASMS) to automatically manage business microservices interoperability at the semantic level and ensure more flexibility and agility.
- We integrate new designs and features from CQRS pattern and other widely standard ontologies, such as (1) Semantic Sensor Network (SSN) [8] (2) SAREF for eHealth Ageing Well domain (SAREF4EHAW) [9] to build a new ontology model called CxS-CQRS (Context-aware Semantic Command Query Responsibility Segregation Ontology) that provide homogeneous data which facilitates the execution of a set of microservices in an efficient way.
- We manage complex user requests by addressing context such as sensor failure and mobility through semantic-based service discovery and atomic request decomposition. Microservices in a multi-fog cloud environment collaborate to ensure responsive care delivery. Previous responses are reused and adapted to optimize performance without recomputing full processes.
- We implement a dynamic semantic-based CQRS broker that provides desired results for ongoing requests of various users while minimizing response time, and maximizing accuracy. We evaluate the proposed approach on pregnant women case study using distributed health organizations in terms of precision and response time.

The research work is structured as follows. Architectural patterns related to data and application lifecycle management are presented in Section 2. Section 3 discusses the relevant literature regarding application management approaches. The ontology model is presented in Section 4. The methodology is detailed under Section 5. Section 6 includes a full analysis and the findings of the case study. Section 7 presents the key findings and limitations. Finally, Section 8 of the paper focuses on conclusion and the future works respectively.

2. Architectural Styles and Patterns: Background and Definitions

This section discusses three main application architectural styles and exposes application lifecycle management using a standard CQRS and event sourcing pattern.

2.1. Monolithic architecture

Monolithic architecture (Figure 1(a)) centralizes all functions and data in a single deployable unit [10], simplifying development. However, as applications grow, it becomes harder to scale and maintain due to tight coupling and performance bottlenecks. Issues include slow deployment, limited fault isolation, and challenges in coordination and database flexibility.

2.2. SOA (Service Oriented Architecture)

SOA (Figure 1(b)) allows the reuse of interoperable and distributed components [11]. Data is exchanged via synchronous or asynchronous messages. It reduces maintenance effort and cost by promoting cooperation between systems through a common communication language.

2.3. Microservice architecture

Microservice architecture (Figure 1(c)) extends SOA, enabling modular, distributed development. Each lightweight service can use its own or shared database and be built with heterogeneous technologies. This flexibility supports scalability and specialization. Major vendors like Amazon, Netflix, and LinkedIn have adopted MSA for efficient, high-performance service delivery [12][13]. Figure 1 presents the different development approaches in general.

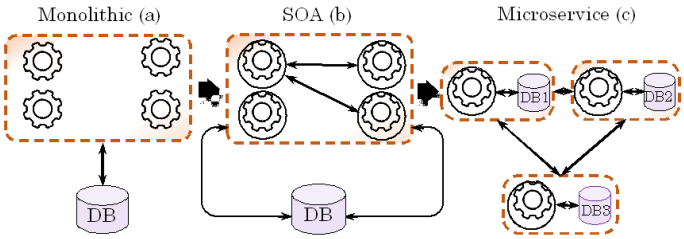


Figure 1. Evolution of different architectural styles

2.4. CQRS and Event Sourcing

CQRS decouples write and read models in microservices [13][14], allowing each to be optimized independently. For example, using materialized views in Elasticsearch or MongoDB for fast queries. Event sourcing records every state change as an immutable event in an Event Store, which is published via an Event Bus. Combining CQRS with event sourcing(Figure 2) ensures consistency by having Event Handlers update read side materialized views from the event stream [15]. This pattern supports efficient complex queries and reliable data synchronization.

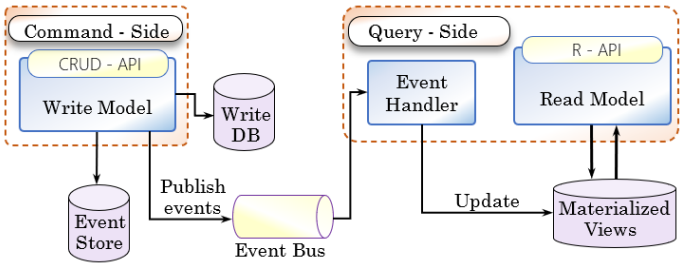


Figure 2. Functioning of CQRS and Event Sourcing

Context aware IoT data management involves tasks, services, and middlewares that adapt to dynamic contexts and uncertainties. Recent platforms fall into monolithic [16–19], microservice based [20–28], and context aware [29–32] categories. However, none fully address semantic interoperability or the demands of smart health crises.

2.5. Monolithic Architecture based Platforms

Shinde et al. [16] introduced SF, a DNN based sensor fusion method for real time COVID 19 detection and GPS triggered alerts, though all data resides in ThingSpeak. Li et al. [17] developed a cloud managed IoT system for at home maternal monitoring, improving continuity and reducing miscarriage risk but it fails offline. Integrating fog computing for local data analysis could ensure real time availability despite Internet outages. Munir et al. [18] proposed a fog deployed IoT system combining an ontology model with KNN to forecast irrigation timing and optimize water use. However, scalability issues caused high water consumption and fog server overload. This highlights the need to offload KNN prediction to the cloud while retaining the lightweight ontology model at the fog. Khan et al. [19] proposed a Warehouse Management System (WMS) based on IoT sensors to speed up information flows between different departments of the company. A prototype was developed and deployed in a textile factory to validate the added value of the system. However, experiments resulted in increased overheads and failed to control remote logistics services.

2.6. Micorservices Architecture-based Platforms

Panchea et al. [20] presented a telemedicine platform called OpenTera based on SOA developed with several frontend and backend technologies. This work used IoT devices and robotics to provide long-term care, especially

for the elderly during COVID-19. OpenTera platform was designed to interact with social robots that offer cognitive services. Taneja et al. [21] developed a fog based microservices IoT platform for real time herd health monitoring and dairy productivity, ensuring operation despite cloud outages. Xu et al. [22] designed a Fog and Kubernetes architecture to automate Industry 4.0 robots and sensors, reducing errors. Both face challenges in running distributed ML and complex queries across multiple fog nodes. Applying CQRS with event sourcing can consolidate data into a unified source for consistent updates. Integrating ontologies further standardizes heterogeneous sensor and robot data for seamless interoperability. Ortiz et al. [23] present a CEP based microservices approach to unify real time IoT streams (JSON, XML, CSV) over AMQP and MQTT in smart ports. Validated with the nITROGEN simulator, it demonstrates robust technical interoperability for intensive data streaming. Mavrogiorgou et al. [24] introduced beHEALTHIER to manage eXtended Health Records using four pillars: Data, Information, Knowledge, and Action. It integrates diverse data sources to support healthcare professionals. However, it lacks architectural patterns like CQRS or SAGA for enhanced data management. Venkatesh et al. [25] proposed a customizable microservice architecture with a recommendation tool based on the CQRS pattern. While effective in guiding data access, it lacks mechanisms for handling service failure and ensuring continuity. Integrating a semantic layer and SAGA pattern is recommended to enhance reliability and transaction consistency. Long [26] enhanced CQRS by integrating it with the SAGA pattern to manage big data and avoid concurrency conflicts. Although effective, the model lacks ontology support for semantic interoperability and load balancing. Including these elements would improve decentralized system performance. Surianarayanan et al. [27] proposed a semantic microservices approach using ontologies for automated discovery based on functional matching. While it supports dynamic service composition, it lacks flexibility and quality in handling nonfunctional needs. Integrating it with the CQRS pattern can enhance agility and dynamicity. Ntentos et al. [28] analyzed 35 patterns for flexible, decentralized microservice architectures and proposed the ADD model covering databases, queries, transactions, and APIs. Their study showed a 21% improvement in decision process design. However, it overlooked combining ontology and CQRS for enhanced semantic and data management.

2.7. Context-aware Services Architecture-based Platforms

Alti et al. [29] introduced Kali-Smart, an autonomous platform for reconfigurable mobile health apps using event-condition-action services for adaptive responses. Ramírez et al. [30] proposed a DSL-based medical event detection approach using AGGIR for assisting disabled persons. Sanaa et al. [31] and Phu et al. [32] developed context-aware systems using ML and edge deployment to optimize smart home and IoT data sharing.Context-aware platforms can benefit from hybrid microservices across fog and cloud to improve scalability and QoS. Integrating CQRS and ontology enables efficient query handling and semantic interoperability across diverse systems.

2.8. Discussions

Table 1 highlights differences in IoT architectures based on semantic data handling, reusability, and dynamicity. Microservices offer better scalability and flexibility than monolithic models, especially in dynamic environments [10–24]. Fog nodes handle fast tasks like sensing, while cloud services process deep analytics such as classification. Ontologies ensure semantic consistency across diverse sensor data from various manufacturers. AI execution, query optimization, and service continuity are enhanced through semantic reasoning and dimensional reduction.

3. CxS-CQRS: Context-aware Semantic CQRS Ontology

This section presents our proposed CxS-CQRS ontology that hides the data sources' heterogeneity of various sources and types as well as the heterogeneity of microservices of several data types (queries, events), and different communication protocols. CxS-CQRS ontology provides new classes and properties, and imports others from SAREF4EHAW, SSN, Time [33] ontologies to provide an expressive view of various users in read/write queries. These ontologies are respectively referenced by the prefixes SAREF4EHAW: ssn: and time. Our goal is to fill gaps of existing ontologies and provide dynamic interoperable microservices with respect to CQRS to manage service in

Table 1. Review of existing IoT Platforms for data management

Related Works	Generality	Reusability	Extensibility	Dynamicity	Sensor Diversity	Mobility	Services Diversity	QoS	Event Flow	Data Flow
Monolithic										
[16]		×				×	×			×
[17]					×	×	×			×
[18]					×		×			×
[19]						×				×
Microservice										
[20]		×		×	×	×	×			×
[21]		×		×		×				×
[22]				×	×	×				×
[23]				×						×
[24]				×	×	×	×			×
[25]				×					×	×
[26]				×					×	×
[27]	×			×						×
[28]									×	×
Context Service										
[29]	×	×	×	×	×		×	×		×
[30]	×	×	×	×	×		×	×		×
[31]				×	×	×				×
[32]	×	×	×	×		×		×		×
Proposed Approach										
	×	×	×	×	×	×	×	×	×	×

the right manner by separating context from data service management and service control. As depicted in Figure 3, the CxS-CQRS ontology consists of two layers:

1. The core layer (bold blue color) which contains generic core concepts for representing CQRS microservices (e.g., microservices events publisher, microservices events consumers), command and query flow read and write (Semantic Query Controller Microservice, Semantic Command Controller Microservice) and QoS (QoS Contract, QoS), data sources and events.
2. The domain-specific layer (light blue color) which enables the alignment of CxS-CQRS ontology with external ontologies to represent the knowledge of specific domains such as smart health, smart transport, smart factory, smart homes, or smart cities. Firstly, we start by detailing the core concepts of CxS-CQRS then we detail its related sub-ontologies.

3.1. CxS-CQRS Core Ontology

The CxS-CQRS core ontology (Figure 4) defines foundational concepts for modeling microservices across domains. Its key concept, the Microservice class, includes metadata such as name, role, version, API type, and output format. Each microservice can be described using OpenAPI, AsyncAPI, or GraphQL and aligned with domain-specific ontologies. This structure supports flexible integration and specialization across domains like health, transport, and business. The CxS-CQRS ontology supports microservices that handle input/output data (events, queries) across various applications. Services are deployed on IoT devices, fog nodes, or cloud environments, and multiple microservices may serve one provider. It aligns with the SAREF4EHAW ontology to represent health-related microservice concepts and relationships.

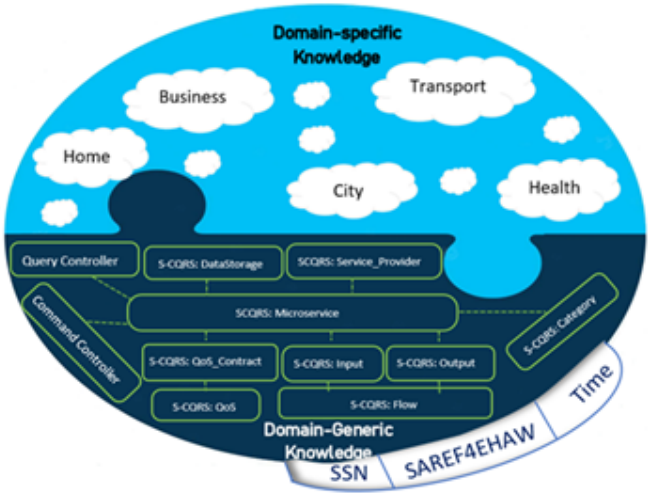


Figure 3. Domain Generic and Domain Specific of CxS-CQRS Ontology

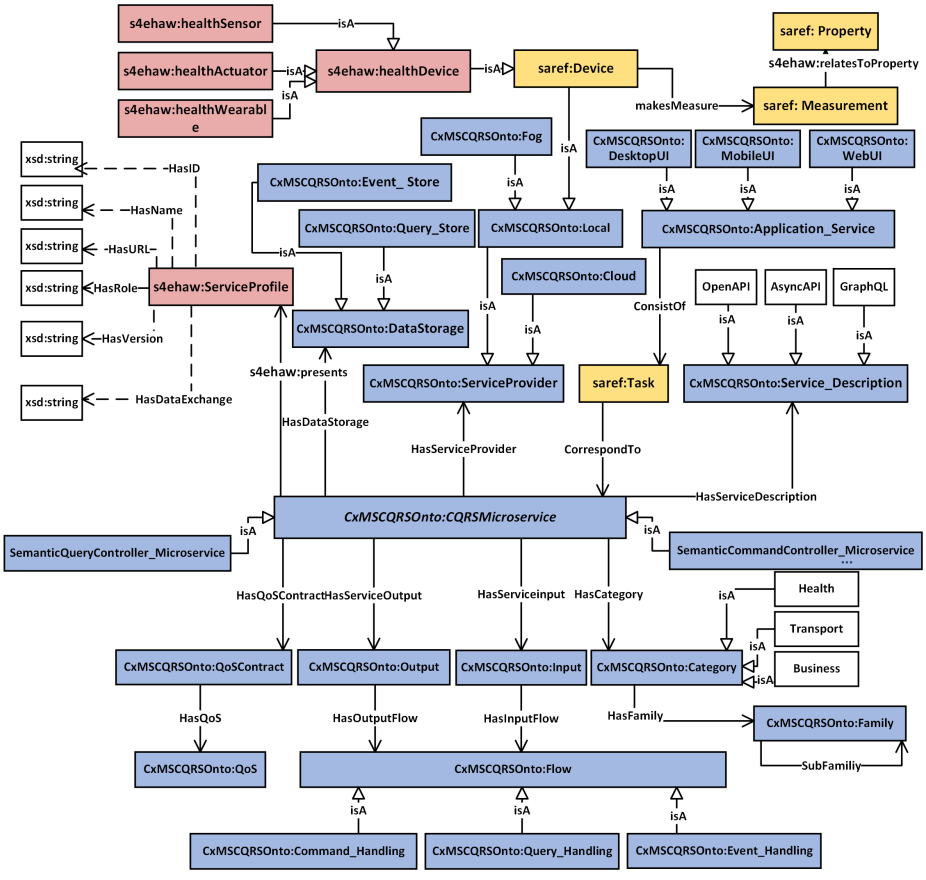


Figure 4. CxS-CQRS Core Ontology

3.2. *Microservices sub-Ontology*

To ensure high scalability and flexibility, CxS-CQRS separates “Read” and “Write” microservices, optimizing queries and adapting to dynamic contexts like sensor failure or new events. This separation introduces two microservice types: Semantic Command Controller and Semantic Query Controller. As illustrated in Figure 5, the Semantic Command Controller manages event publishing and calls the Semantic Command Handler to determine which events to send to the EventBus. It handles user commands, validates requests, and updates databases with CUD operations. The Semantic Query Controller retrieves data from the EventBus via the Semantic Event Consumer and ensures synchronization. Query results are then published to the Query Bus for timely and accurate responses.

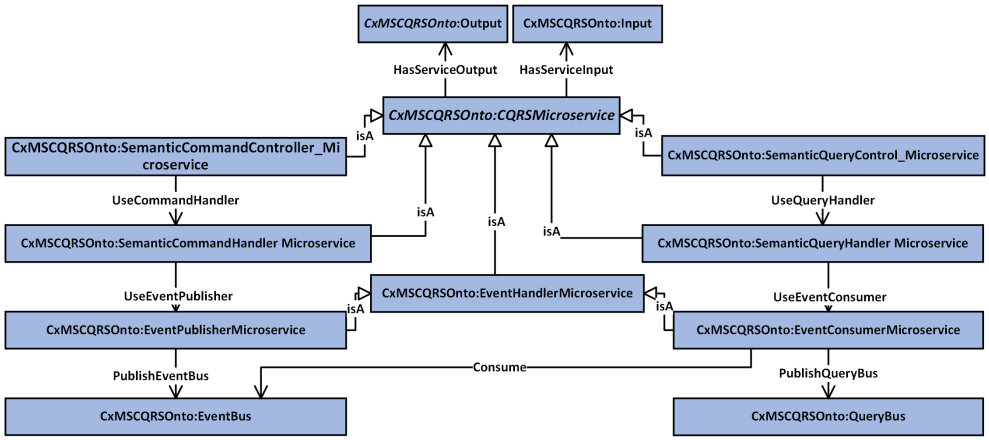


Figure 5. Microservices sub-Ontology of CxS-CQRS

3.3. *Flow Sub-Ontology*

As illustrated in Figure 6, we describe data flow using CxS-CQRS: Flow concept to demonstrate and transmit a set of CxS-CQRS Microservice inputs/outputs messages to other microservices. Each CxS-CQRS: Flow manipulates domain-specific ontology objects and concepts (for example, SAREF4EHAW) to use clear and understandable terms.

We define three CxS-CQRS flows: Event Handling, Command Handling, and Query Handling.Query Handling translates application-level queries into executable forms using specific operations and conditions. Event Handling manages how events tied to domain entities are published, consumed, these events carry contextual information like time, location, and system unit, and can be transformed into semantic Data. Command Handling involves Create, Update, and Delete (CUD) operations on domain entities.

3.4. *QoS Sub-Ontology*

CxS-CQRS microservice can have different levels of QoS corresponding to various functionalities represented in the CxS-CQRS ontology (Figure 7). We also define a group of equivalent microservices that perform the same functions but have different QoS values that correspond to one of three semantic levels (high, medium, low). The term QoS refers to a set of metadata parameters. These parameters include security, time, latency, reliability, price, . . . etc.

3.5. *Context Sub-Ontology*

We aim to separate context from data and service management views to ensure high-quality services at the right time and location. As illustrated in Figure 8, location and time are key contextual concepts in managing entities

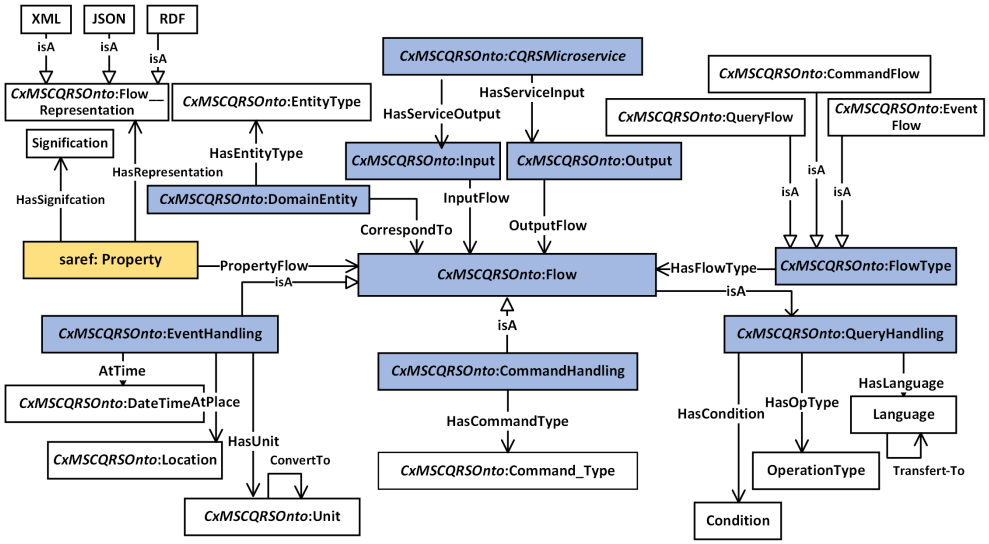


Figure 6. Flow sub-Ontology of CxS-CQRS

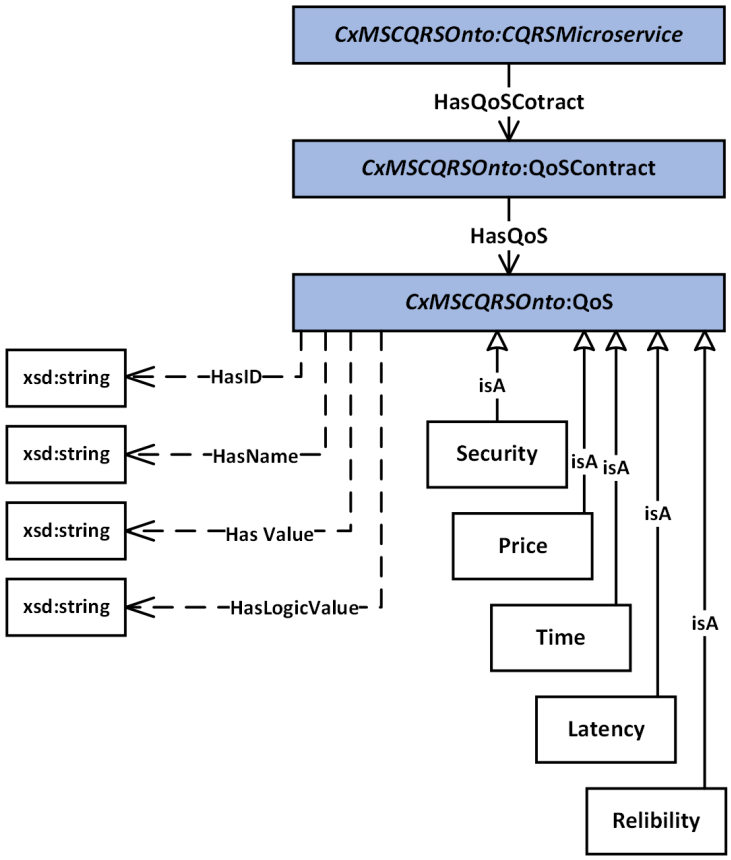


Figure 7. QoS sub-Ontology in CxS-CQRS

including patient, device, and activity. Each entity has its proper contextual properties. Some properties track the location of the patient, the health status of that patient, nearby monitoring devices, and services accessibility are also considered. However, we scope service accessibility through space-time constraints. High levels of service accessibility as a result of excellent services which considers reliable and high service qualities.

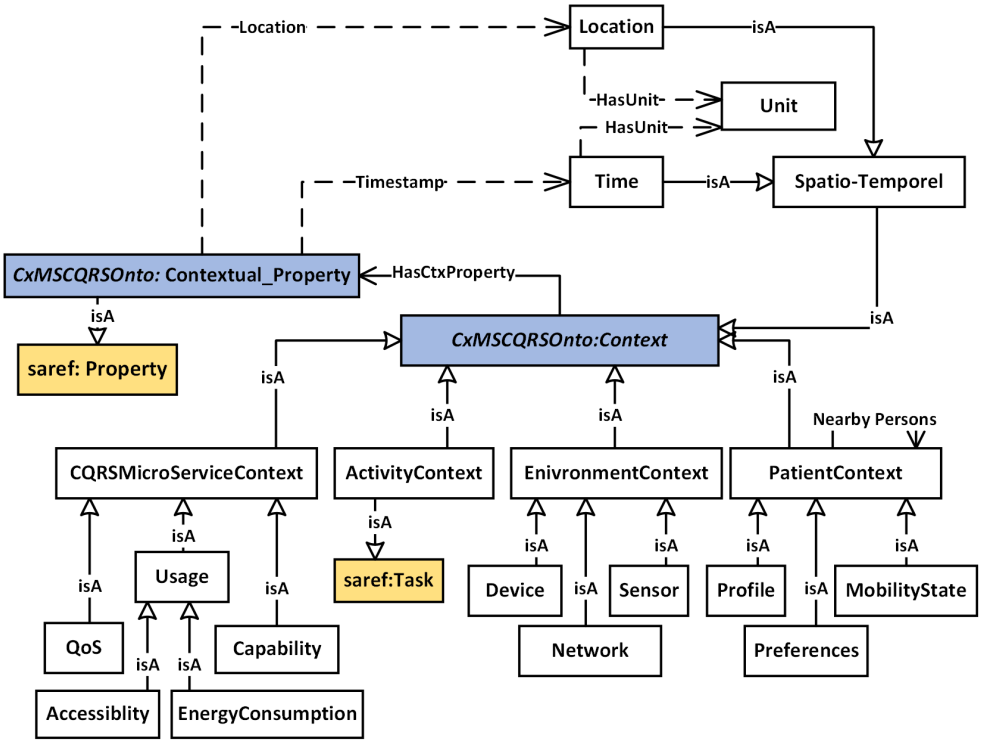


Figure 8. Context sub-Ontology in CxS-CQRS

4. DS-CQRS: Dynamic Semantic-Based Microservice CQRS Management Framework

In health field, context management is an essential aspect of smart personal care. Huge number of requests can be posted by users at the same time under the evolution of patient's context and it is a big challenge to manage dynamic context changes and en-sure continuous service delivery. Therefore, for this purpose, a framework called DS-CQRS (Dynamic Semantic-based Command Query Responsibility Segregation Pattern) is required which provides more flexibility and efficiency in managing data through two sides: read and write. It extends CQRS by a new semantic layer called "Semantic Event Bus" and provides a clear separation between semantic command microservices and semantic query microservices over distributed healthcare data sources. Two key aspects are defined on DS-CQRS: dynamic and semantic. With dynamic aspects, service continuity under different circumstances is ensured. With aspect of se-mantic microservice, interoperability is provided while retaining flexible and extensible data structures. We begin by describing the DS-CQRS agents. We present the functional model of the proposed framework. Then we go over the microservice matching, microservices orchestration quality, microservices selection, and execution algorithms that were used in this research work.

4.1. General Architecture

The proposed knowledge-based context-aware CQRS framework is given in Figure 9 which consists of four main layers as described below:

- User Interface (Application client): allows the user to express both their requests and commands and receive expected results. It can be either Mobile UI, Desktop UI, or Web UI.
- API/Gateway: acts as a proxy for routing all UI incoming queries and commands and exposes API microservices (REST API, SOAP API) in front of event-driven integration. The API Gateway as an intermediate layer between a user and a platform.
- Platform: is dynamic semanticbased event-driven and offers the semantic treatment queries and commands over various and distributed data sources (e.g., databases, sensors, documents, external resources), as well as a dynamic microservices orchestration strategy based on incoming events (e.g., mobility of the user, sensor's failure, context changes). To publish/consume events, all microservices communicate via a semantic event bus. It is made up of the following agents:

  1. Context-aware Semantic Microservice CQRS Broker: represents a semantic mediator between a user's application and different data sources based microservices which consist of six types of agents that interact and collaborate to realize and manage all microservices discovery, request matching, and microservices orchestration process. When accessing the knowledge base, each agent plays a specific role. The following is a description of all agents in the semantic Microservice CQRS broker:
     - The Semantic Microservice Discovery Agent for discovering atomic, loosely coupled and collaborated event-driven microservices with respect to user requests and user commands.
     - The Semantic Microservice Matcher Agent is a semantic matching engine responsible for generating a global CQRS microservices ontology of one or more ontologies of local microservices and producing interoperable semantic microservices while accessing different data sources.
     - The Semantic Microservice Orchestrator Agent uses semantic descriptions to discover, configure, and coordinate microservices based on context. It ensures seamless interaction and data extraction from diverse sources, enabling flexible and scalable information processing.
     - The Semantic Microservice Manager Agent adds/updates/remove one or more semantic-based event-driven microservices applications.
     - The Semantic Command Microservice Agent manages write-side operations, such as data creation, updates, and deletion by leveraging a global ontology that semantically describes available microservices. This ontology enables the agent to reason about service capabilities, ensuring context-aware selection and execution of appropriate commands. Through semantic alignment, the agent dynamically coordinates interoperable services across diverse platforms, supporting conflict resolution, data validation, and policy enforcement to enhance system robustness and adaptability.
     - The Semantic Query Microservice Agent is responsible for handling data retrieval (read-side) operations in a distributed, semantically enhanced microservice architecture. It operates by utilizing a global ontology, which serves as a unified semantic representation of the system's domain concepts, relationships, and available data services. When a query request is received typically formulated in terms of high-level domain concepts rather than low-level technical specifications. The Semantic Query Microservice Agent consults the global ontology to interpret the meaning of the request. It then identifies and orchestrates the appropriate set of microservices needed to fulfill the query, ensuring that the response is accurate, relevant, and semantically consistent.
  2. Registries: contains various kinds of CQRS microservices, including services published by hospitals, laboratories, clinics, etc. It also includes events related to domain-centric knowledge (standard health ontology).
  3. Semantic Event Bus: it serves as a mediator between semantic microservice CQRS broker and different microservices and stores (e.g., event store and query store).

- IoT sensors and actuators: it contains a list of sensors that collect data and sense events on locations that are transmitted and stored on the event processing stream.

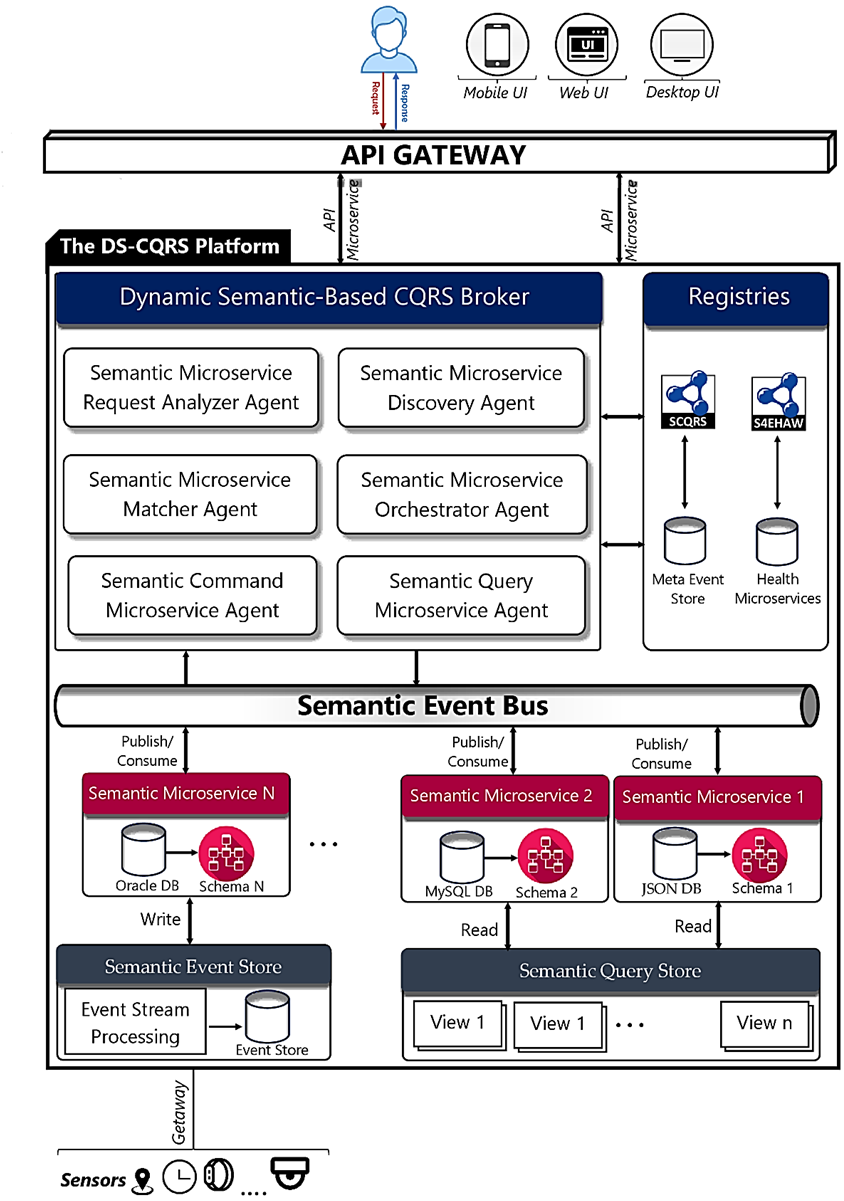


Figure 9. The proposed DS-CQRS platform

4.2. Functional Model

The process begins with the reception of a user request. For instance, a physician queries a list of patients who have recorded high ECG measurements in Montreal city on the current day. This request encapsulates both health-related parameters and contex-tual constraints. Key contextual elements such as the patient's health status and geo-graphic location are essential for filtering relevant CQRS-based microservices. To handle this, we employ a Semantic

Request Analyzer, invoked by the Semantic Microservice CQRS Broker, which decomposes the user request into a set of simpler sub-requests. This decomposition offers multiple benefits, including reduced complexity and improved microservice discoverability.
For the discovery process, a semantic context-aware matching model is applied over semantically enriched service registries. The Semantic Microservice Matcher agent runs a matching algorithm that maps user-defined sub-requests to the semantic de-scriptions of available microservices. These sub-requests may involve retrieving patients with elevated ECG values, filtering data to include only records from the current day, limiting results to patients residing in Montreal, and isolating those under the care of the requesting physician. The matching process leverages domain-specific ontologies such as CxS-CQRS and SAREF4EHAW to interpret semantic annotations and refine the selection of microservices according to contextual parameters and Quality of Service (QoS) requirements. At this point, two operational scenarios can be distinguished:

- If a chain of avaliable semantic microservices that can answer the user's request, the Semantic Microservice Orchestrator orchestrates these microservices to retrieve the requested information from various distributed views and return it to the user. When a case of user request is the command type, the Semantic Microservice Orchestrator orchestrates and writing microservice and publishes events in a bus to synchronize the read side.
- If the Semantic Microservice Matcher agent does not find any match, it sends back "any results" to the user. We can see that the newly discovered microservices are stored in the ontology model and linked to their health data source using the Semantic Microservice Manager agent to improve the application's functionality. The functional model of DS-CQRS platform is illustrated in Figure 10.

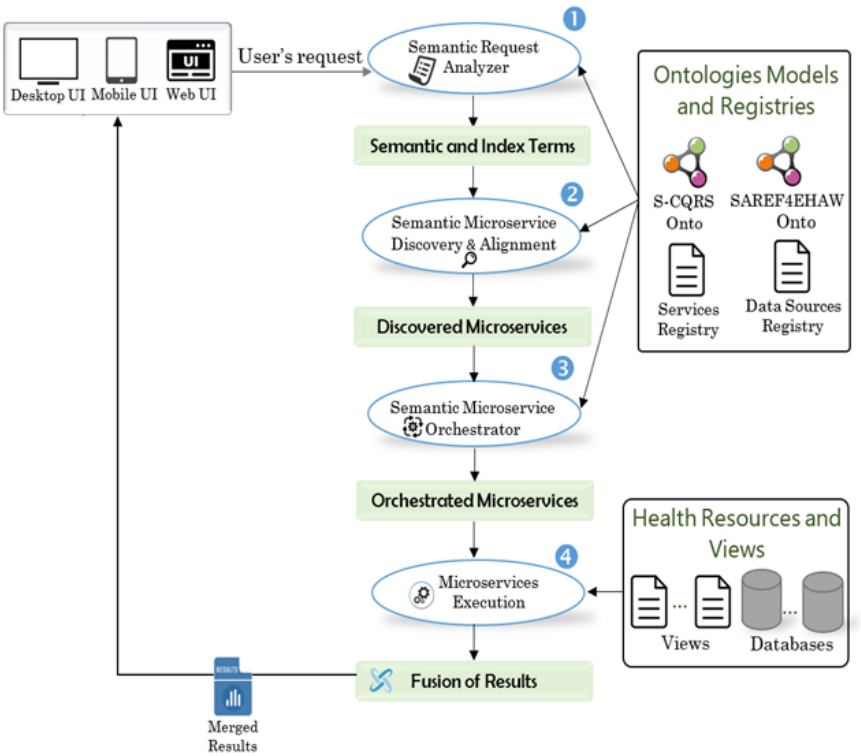


Figure 10. Functional model of DS-CQRS framework

4.3. Security and Privacy

Given the sensitivity of healthcare data, the proposed framework integrates several security mechanisms to ensure confidentiality, integrity, and access control across distributed microservices. All medical data are protected via TLS-based secure communication during transmission and AES-based symmetric encryption technique during storage of data, ensuring end-to-end data protection. Authentication and authorization are enforced through secure API gateways using token-based mechanisms (e.g., OAuth2/JWT), allowing only authorized entities to access patient data.
To enhance security, we integrate a context-aware Ciphertext-Policy Attribute-Based Encryption (CP-ABE) mechanism combined with blockchain-based trust mechanisms inspired by Annane et al. [35]. This mechanism enables fine-grained, context-dependent access control, where data decryption is permitted only if both user attributes and contextual conditions are satisfied. The integration of blockchain additionally ensures data integrity, traceability, and decentralized trust among distributed microservices. Moreover, the semantic CQRS broker enforces context-aware access control policies based on user roles and context, ensuring that only trusted and authorized microservices participate in service discovery and orchestration. Finally, compliance with privacy and data governance requirements such as GDPR is also considered in the system design for healthcare data protection regulations.

4.4. Detailed Algorithms

Dynamic semantic Context-aware CQRS process involves four main phases: (1) input request/ semantic pre-processing, (2) context-aware quality semantic micro-services filtering engine (3) microservices orchestration and (4) microservices execution engine.

4.4.1. Input Semantic Request and Pre-Processing A user submits his request (e.g., read query, write command) with service qualities through the application UI. The Semantic Microservice CQRS Broker is responsible to communicate with the corresponding microservices to start the semantic query pre-processing and then returning the results. The Semantic Request Analyzer agent breaks down the user's request into simple requests (e.g., simple read queries or simple write commands). Each simple request is decomposed into a set of health information (e.g., health terms). The information in a simple request is semantically described in SAREF4EHAW health ontology. After that, the WordNet-based lexical and semantic similarities are computed between simple request terms and ontology concepts. Finally, using the SAREF4EHAW health ontology, semantic ordered relationships among concepts of simple queries are elaborated and matched.

Algorithm 1: Semantic Request Analyzer

Input: request req, domain ontology O_{domain}, CxS-CQRS ontology $O_{CxS\text{-}CQRS}$
Output: semantic sub-requests SR, context Ctx, QoS preferences QoS_{pref}

Step 1: Initialization

$SR \leftarrow \emptyset, \quad Ctx \leftarrow \emptyset, \quad QoS_{pref} \leftarrow \emptyset$

Step 2: Request Parsing

Parse req to extract request type (Read / Write), domain concepts, spatio-temporal and QoS preferences.

Step 3: Normalization Normalize all extracted terms

Step 4: Semantic Mapping Map extracted terms to ontology concepts using O_{domain} and $O_{CxS\text{-}CQRS}$

Step 5: Ambiguity Resolution

if a term is ambiguous **then**
Resolve using ontology relations and contextual information

end if

Step 6: Context Extraction

Extract context constraints:location, time, patient condition, device status, mobility and emergency state

Step 7: QoS Extraction

Extract QoS preferences:latency, reliability, security and cost

Step 8: Decomposition

Decompose req into semantic sub-requests:

$$SR = \{sr_1, sr_2, \ldots, sr_n\}$$

such that each sr_i represents an atomic task

Step 9: Annotation

for each $sr_i \in SR$ **do**
– attach semantic concepts
– assign request type
– associate context constraints
– assign QoS preferences
end for

Return: SR, Ctx, QoS_{pref}

4.4.2. Context-Aware Quality Semantic Microservices Filtering We identify the relevant event-driven microservices and data-related microservices that match simple queries (e.g., simple commands) of the user that have to be managed by the platform and will enrich our CxS-CQRS ontology model. Based on user simple queries and semantic services information (e.g., microservice category, microservice role, QoS, microservice inputs/outputs, etc.) obtained from shared data sources, the discovery process has been automated by a Quality Semantic Matching Microservice Discovery. The idea of matching between the microservices and the user's simple queries is to determine how similar two concepts, two categories, two terms, and QoS factors based on health thesaurus, CxS-CQRS ontology, and standard SAREF4EHAW health ontology. Semantic microservices are classified by category (e.g., pregnancy, diabetes, anemia, COVID-19, etc.), role (e.g., monitoring, analysis, treatment, danger), QoS, and defined by their contextual attributes (e.g., contextual events, context data type, etc. It may also be dependent on other atomic microservices that define some inter-microservice relationships. So, we combine two similarity measures: the first is semantic similarity, which measures the relationship between a user's simple request and microservice concepts, and the second is context similarity, which measures the context of a patient and the contextual attributes of microservices. Microservice description and user's semantic request are quite similar thanks to the following measure:

$$Sim(q, s) = W_{Sem} \times Sim_{Sem} + W_{QoS} \times Sim_{QoS} + W_{ctx} \times Sim_{ctx} \tag{1}$$

where W_{Sem}, W_{QoS}, and W_{ctx} are weights in the range $[0, 1]$ that indicate the relative importance of semantic similarity (Sim_{Sem}), quality similarity (Sim_{QoS}), and contextual similarity (Sim_{ctx}), A score that exceeds a given threshold (e.g., 0.8) means good similarity between the user queries and context-aware microservices concepts. The semantic similarity is based on Wu-Palmer measure[34] and calculated using the following formula:

$$Sim_{Sem}(q, s) = wup(q_{c_1}, s_{c_2}) = \frac{2 \cdot dist(c)}{dist(q_{c_1}, c) + dist(s_{c_2}, c) + 2 \cdot dist(c)} \tag{2}$$

where c is the most specific concept that includes the concepts c_1 of request q and c_2 of service s, and $dist(c_i, c)$ is the number of edges between a concept c and another concept c_i.

The quality similarity measure is used to ensure maximum benefit by aligning the user's QoS preferences with the QoS parameters of the services. This measure is evaluated as follows:

$$Sim_{QoS}(q,s) = \sum_{i=1}^{nb_{QoS\_Params}} W_u \times \left(\frac{q^{qos_i} - s^{qos_i}}{q^{qos_i}} \right) \tag{3}$$

where a required quality of service q^{qos_i} will be matched with the provided quality of service s^{qos_i}. q^{qos_i} and s^{qos_i} vary between 0 and 1. W_u is the user weight for given QoS. The contextual similarity measure is used to ensure context matching based on patient's context and service's contextual parameters and values. This measure is defined as follows:

$$Sim_{ctx}(q,s) = \sum_{i=1}^{nb_{ctx\_Params}} \left(\frac{q^{ctx_i} - s^{ctx_i}}{q^{ctx_i}} \right) \tag{4}$$

Algorithm 2: Context-Aware Semantic Microservice Selection

Input: User request q_u, Microservice set S, Ontology (CxS-CQRS / SAREF4EHAW), Context data q_{ctx}, QoS attributes, Weights W, Top-N value

Output: Ranked list of Top-N recommended microservices

Step 1: Semantic-Based Attribute Similarity

– Transform the user request q_u into a concept vector.
– Transform each microservice S_j into a concept vector.
– Define the weight vector $W = (w_1, w_2, ..., w_k)$,
where weights correspond to service name, service role, inputs/outputs, service size, and service category.
– Compute semantic similarity for each service using the ontology-based Wu-Palmer measure (Eq. 1).
– Generate the semantic candidate set.

Step 2: Context-Based Similarity

Step 2.1: Context Normalization

– Normalize user and service context values using min-max normalization.

Step 2.2: Matching Matrix Construction

– Construct the matching matrix R:
If historical context exists: compute real similarity scores.
Else (cold-start): initialize $R_{ij} \in \{0, 1\}$.

Step 2.3: Context Similarity Calculation

– Compute context similarity using Eq. 3.
– Generate the context candidate set.

Step 3: QoS-Based Similarity – Normalize QoS attributes using min-max normalization.
– Compute QoS similarity using Eq. 4.
– Generate the QoS candidate set.

Step 4: Final Aggregation and Ranking

– Compute the total score for each service using Eq. 1.
– Sort all services in descending order of $Score(S_j)$.

– Select the Top-N services.
Return:the ranked Top-N services

The *Quality Semantic Microservice Updater* agent can choose a suitable microservice from the filtered equivalent services list MS_{out} within the same functionalities based on the sensor's availability in the environment (e.g., low battery), the appearance of a new sensor (e.g., a new type of event), and the availability of the microservice provider (e.g., server unavailable). A set of context changes will be notified to the *Quality Semantic Matching Microservice Discovery* to dynamically add a newly discovered microservice in the ontology model, and remove an unavailable microservice from the ontology. Finally, the *Microservice Orchestrator* agent receives an updated list of microservices for orchestration.

4.4.3. Semantic Microservices Orchestration We generate a chain of microservices by using semantic dependencies among distributed microservices that associate available data sources/views and ordered relationships among simple user queries. Each simple query corresponds to an atomic microservice (read/write). Atomic microservices are orchestrated as ordered relationships among simple queries into a process using Semantic Microservices Orchestrator and thus act together as a single composite service to respond user's query (user's command). Using CxS-CQRS ontology reasoning, the orchestrator ensures compatibility between service inputs and outputs, correct execution order and dependency resolution between services to perform dynamic and automatic composition without manual configuration.

4.4.4. Semantic Microservices Execution Engine We have here two cases:
• If the user request corresponds to write commands, then the Semantic Microservice Command launches distributed event-driven microservices to publish events to the event store and synchronize them with various associated views and data sources.
• If the user request corresponds to read query, then the Semantic Microservice Query invokes distributed data microservices to retrieve data and publish it to an event store to be consumed by other linked data microservices.

Algorithm 3: Microservices Orchestration and Execution
Input: Semantic sub-requests SR, MatchSet, ontology $O_{CxS-CQRS}$, EventBus
Output: Ranked list of Top-N recommended microservices

Step 1: Initialization
$Workflow \leftarrow$ empty directed graph

Step 2: Service Selection
for each $sr \in SR$ **do**
Select best service m_{best} from $MatchSet[sr]$ using Algo.2
Add m_{best} as a node in $Workflow$
end for

Step 3: Dependency Construction
for each pair (m_i, m_j) in selected services **do**
Check semantic dependency using $O_{CxS-CQRS}$:
– output(m_i) matches input(m_j)
– compatible domain concepts
– compatible request type and context
if dependency exists **then**
Add edge $m_i \rightarrow m_j$ to $Workflow$
end if
end for

Step 4: Workflow Validation
Check:no missing inputs, no incompatible input/output, and no invalid cycles
if validation fails **then**
Replace invalid service with next candidate from $MatchSet$
Recompute dependencies
end if

Step 5: Execution
Deploy selected microservices
Execute $Workflow$ in dependency order
Publish events via EventBus
if request type = Write **then** update CQRS read model
if request type = Read **then** return query results

Step 6: Dynamic Adaptation
Monitor context changes during execution
if service failure or context change occurs **then**
Trigger re-matching
Substitute with equivalent service
Update $Workflow$ dynamically
end if

Return: $Workflow$

5. Experimental Results and Discussion

In this section, we first present the prototype development, then the dataset used to evaluate the proposed approach, the experimental setup, and finally the analysis.

5.1. Prototype Development and Experimental Setup

A prototype is developed using a Docker-based microservices architecture that focuses on six main agents: Discovery Agent, Matcher Agent, Orchestrator Agent, QoS Manager, Command Handler, and Query Handler. These agents collaboratively manage user requests, semantic matching, orchestration, and QoS-aware service selection. We used the edge-fog-Cloud nodes to simulate a real-world distributed environment that was deployed on AWS infrastructure. Particularly, three AWS t3.xlarge instances are used to represent edge/fog nodes responsible for data preprocessing, context handling, and low-latency semantic matching. A more powerful AWS p3.2xlarge instance equipped with an NVIDIA GPU is used to simulate the cloud layer, where global orchestration, ontology management, and computationally intensive tasks are executed. Edge nodes (UAVs/IoT gateways) connected to fog nodes with 10 ms latency.y. We implemented the proposed platform using Java-based Netbeans framework. We used RLlib 2.4.0 for the agent framework, the knowledge base was based on Neo4j, and Apache Kafka was used as a message broker. We managed container orchestration and inter-service communication using Docker and Docker Compose, enabling scalable deployment and isolation between microservices. Each agent communicates through lightweight REST APIs and an event-driven messaging mechanism to ensure loose coupling and asynchronous processing.
The distributed simulation was conducted using the configuration illustrated in Table 2. We compare our approach with the SAREF4EHAW ontology, the semantic microservices of Surianarayanan et al. [27], and standard CQRS.[13], and the proposed DS-CQRS approach, respectively. We present the evaluation with metrics such as initial response time, processing time, and communication time. We also capture incoming events using parallel reading/writing microservices to reduce the response time. We consider the user's needs and preferences under context changes (e.g., the evolution of health status, sensor failure, etc.) to filter relevant microservices and enhance the monitoring and reporting of the patient's situation. To show the effectiveness, we also provide

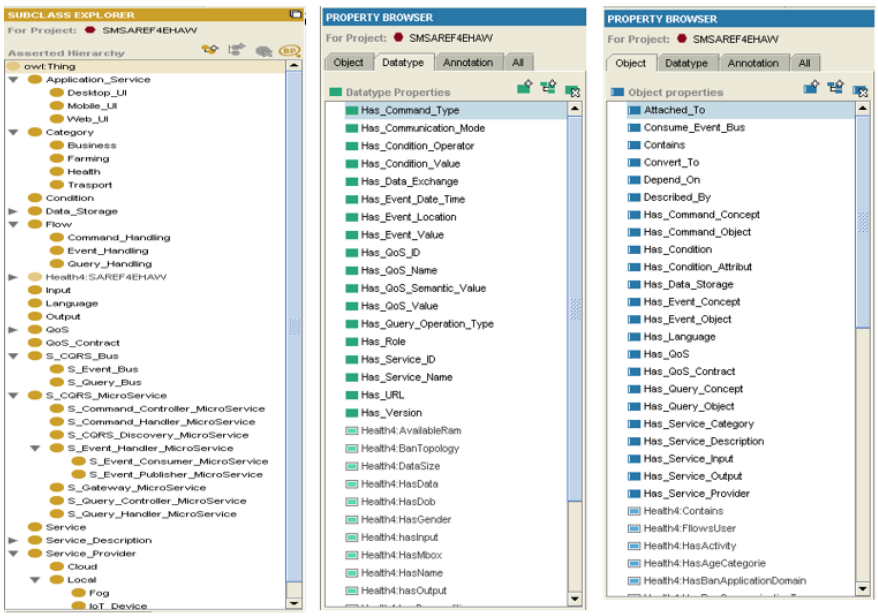
the average accuracy during service discovery and selection, recall, and F2-score evaluated with AWS CloudWatch.

Table 2. Parameter Settings

Parameter	Value
Number of nodes	10 nodes
Microservices	50, 100, 500, 1000
Containers per node	5 – 20
Request rate	50 – 300 requests/sec
Network latency	10 ms (edge–fog), 30 ms (fog–cloud)
Dataset	Healthcare service requests
Deployment tool	Docker + Docker Compose
Communication	REST + lightweight event bus

In our study, we used the Protege tool to implement the proposed CxS-CQRS ontology combined with the SAREF4EHAW ontology as shown in Figure 11. The main functionalities of our prototype are:

- Displaying responses to read/write requests,
- Adding successively new microservices of the health domain,
- Ranking of relevant microservices,
- Ensuring continuity of service by switching microservice by another equivalent and managing medical sensor's failure,
- Managing different data sent or received from heterogenous sensors or external data providers,
- Displaying evaluation reports in terms of accuracy and execution time.



(a) (b) (c)

Figure 11. Implementation of CxS-CQRS ontology using protégé (a) CxS-CQRS classes (b) S-CQRS data properties (c) CxS-CQRS object properties

Each request follows these steps: (1) sensor data is produced at the edge, (2) the local gateway packages the data as a semantic event, (3) the fog node performs semantic matching and checks service availability; if a matching

service is unavailable, an equivalent microservice is dynamically selected, and (4) the cloud node updates the global registry and synchronizes CQRS read/write models. Finally, the response is returned to the user through the optimized query path.

5.2. Dataset Description and Preprocessing

For the realization of the proposed DS-CQRS platform for sensitive applications, we used an experimental dataset of IoT-based medical scenarios of Setif City hospital. This dataset includes 5000 microservices categorized by disease type (e.g., pregnancy, diabetes, anemia, COVID-19, etc.). We also include health core parameters of patients such as age, condition, and mobility, sensor data such as ECG, glucose level, temperature, SpO2, environmental context like location, time, activity, and QoS requirements such as latency, reliability, priority. Each microservice is described using many attributes (name, category, role, version, URL) and QoS parameters (response time, availability, price, security level, reliability). To aggregate the data from different sources, we perform a temporal alignment with timestamps, a standardization of the coordinate system and feature engineering.

We used a probabilistic simulation model to reflect real-world healthcare scenarios. Specifically based on realistic healthcare scenarios, including patient mobility, sensor failures, and varying QoS requirements. Sensor data is generated using Gaussian and log-normal distributions, service request arrival is modeled using a Poisson process to simulate real-time user demand, patient mobility and condition changes are modeled using Markov transitions, and QoS attributes are randomly assigned within realistic bounded ranges derived from typical IoT healthcare systems. Each request is associated with a ground truth service set, and experiments are repeated 10 times with statistical reporting to ensure reliability.

5.3. Evalaution Metrics

We evaluate the effectiveness of DS-CQRS framework in automatically requesting services from varied data set sizes: 200, 400, 600, 800, and 1000 in terms of:

- **Accuracy:** Accuracy measures the overall correctness of the service matching process:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

  where TP denotes correctly matched services, FP incorrectly matched services, FN missed relevant services, and TN correctly rejected services.
- **Recall:** Recall evaluates the ability of the system to retrieve all relevant services:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

  A high recall indicates that the system successfully identifies most relevant services.
- **Precision:** Precision measures the proportion of retrieved services that are actually relevant:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

  In semantic service discovery, high precision ensures that unnecessary or incorrect services are not included in the final composition.
- **F2-score:** The F2-score emphasizes recall, which is critical in healthcare systems:

$$F_2 = \frac{5 \times (\text{Precision} \times \text{Recall})}{4 \times \text{Precision} + \text{Recall}} \tag{4}$$

- **Response Time:** Response time measures the total time required by the system to process a request and return a result:

$$\text{Response Time} = T_{\text{processing}} + T_{\text{network}} + T_{\text{matching}} \tag{5}$$

Response time includes data transmission delay (edge–fog–cloud communication), semantic matching time, orchestration, and execution time. A lower response time indicates faster system responsiveness and better suitability for real-time healthcare applications. In this work, response time is critical because timely decision-making (e.g., anomaly detection or alert triggering) is essential for patient safety.

5.4. Case study: A Dynamic IoT-based Pregnancy Women Monitoring Application

To demonstrate the proposed dynamic semantic-based CQRS platform and its functionalities, the proposed case study is a dynamic IoT-based pregnancy application. Because the daily monitoring of blood pressure, temperature, glucose level, and ECG may be very crucial, the application is intended to dynamically manage IoT-distributed health services and to ensure the regular monitoring of various health data. Also, the application is based on a dynamic semantic-based CQRS platform to avoid the diabetes risk for a pregnant woman, especially in the context of COVID-19 and its critical con-sequences on the baby and her mother by enhancing the monitoring services and the reporting of diabetic pregnant woman's urgent health situations, everywhere and at any time.

5.4.1. Presentation of Case Study and Possible Scenarios The IoT-based pregnancy care application consists in connecting things and de-ploying various health services. A pregnant woman is equipped with wearable bio-sensors (temperature sensor, Glucose meter, Blood pressure sensor, SpO2 sensor, ECG sensor, smartwatch, and GPS tracker) that provide various medical data. These data are collected from heterogenous medical sensors. These medical sensors are used to provide scalar medical data and multimedia contents of the pregnant woman. Various micro-services are deployed in distributed health environments that consumed medical data from their local databases. The monitored information was collected in Gateway. The aggregated context health data of patients in a smart environment are transmitted to different health organizations (hospitals, analysis laboratories, pharmacies, doctor's offices, etc.) through Semantic Microservice CQRS Broker for health data analysis and treatment suggestions. The Semantic Microservice CQRS Broker triggers an immediate response to the treatment device that gives the order to launch different actions such as: sending recommendations by email or SMS, insulin injection, or indicating an emer-gency to the ambulance. The doctor can monitor patient data in real-time thus family members of the pregnant woman can know her localization.

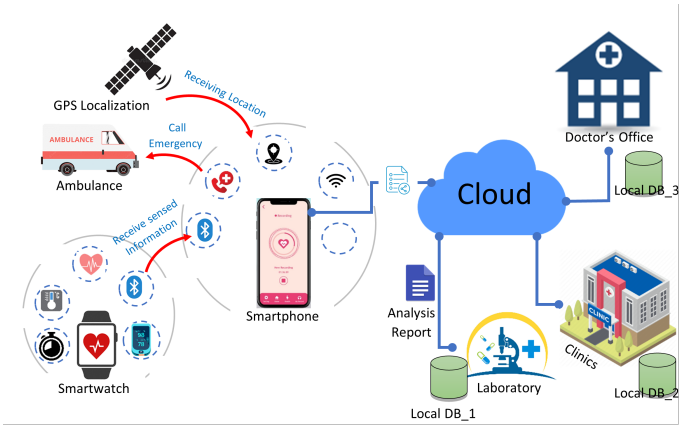


Figure 12. The IoT-based Pregnancy Women Monitoring Application

To ensure continuous healthcare application functioning and real-time health monitoring of a pregnant woman, the platform must monitor the availability of microservices in all situations and must ensure the dynamic processing of requests based on two main events: changes in the sensed data or the discovery of a new microservices (which may contain new stream health data). Whether for woman's healthcare, examinations or treatments, hospitalization, sensor failure, network failure, etc. the doctor needs to keep accessing all his patient's information. The application can take advantage of dynamic semantic-based selection and orchestration of semantic CQRS microservices to

offer quality health services and enhance response time by separating read/write microservices and classifying them by role, category, and QoS. Figure 12 shows the IoT-based Pregnancy Application within DS-CQRS platform.

5.4.2. Illustration of a Case Study Using Prototype To show how our prototype works, we illustrate its application on a pregnant woman of the health domain. The physics specifics his requests. For example, the physics specifies visually the request using query GUI that selects pregnant women with high values of temperature and ECG in SETIF city this week as shown in Figure 13. Then, the physics appreciates faster services as shown in Figure 14. Finally, the physics finds the list of relevant microservices that matched his request and preferences as shown in Figure 15. He can also display the detailed temperature results of a pregnant woman as shown in Figure 16.

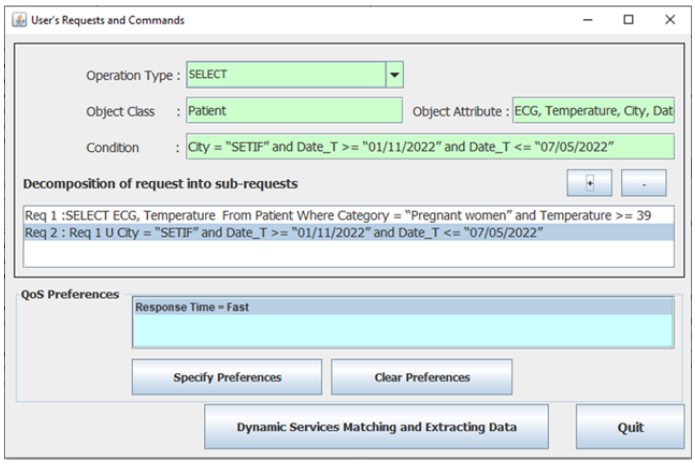


Figure 13. specification of user's requests

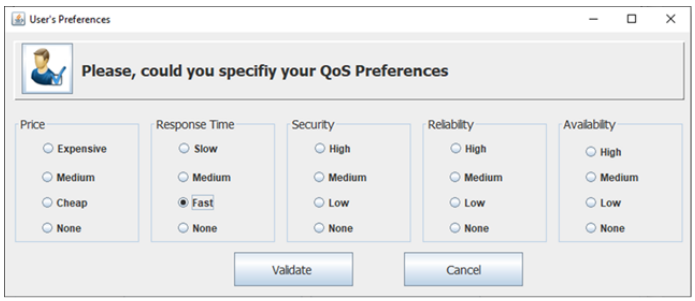


Figure 14. specification of user's preferences

5.5. Possible scenarios

The first scenario is when, the user (e.g., doctor, nurse, analyst) specifies his request as read/write commands and his QoS preferences using GUI. For example, physics wants a list of pregnant women with high values of temperature and ECG results in SETIF city this week in a fast manner. He can also update the evaluation results of specific pregnant women at a specific Date and Time. The semantic CQRS Broker as mediator will receive the user's request from GUI and find appropriate microservices. In this context, the analysis and execution of a user's query/commands involve four main steps:

Step 1. Complex Query Analysis. The Semantic Request Analyzer agent is responsible for analyzing users' queries. This analysis is used to check the syntax and semantics. We notice that to ensure dynamicity, if the user changes his query or changes his health status, a ChangeRequestEvent event is generated. The "Semantic

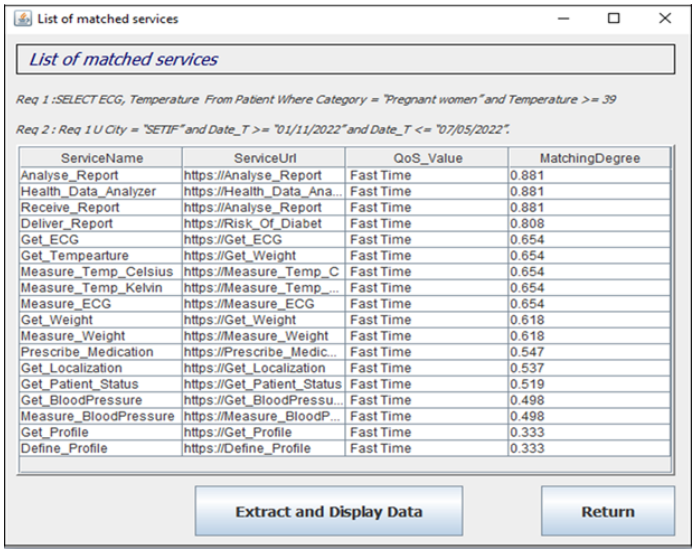


Figure 15. displaying temperature data

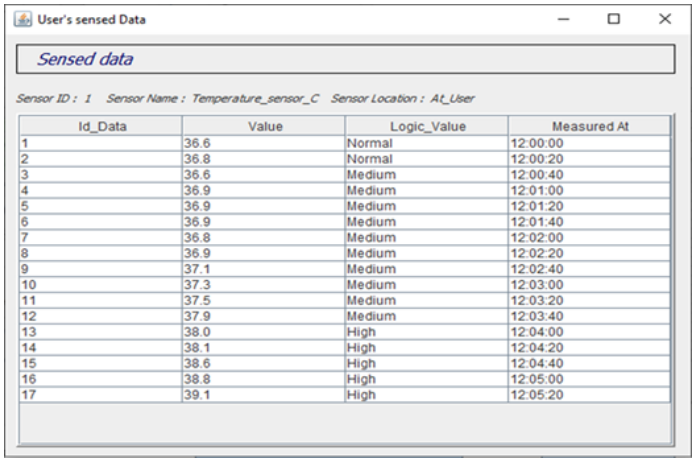


Figure 16. displaying temperature data

Request Analyzer agent receives and extracts the location/time operator and different Join operators. Table 3 present An example of the decomposition of a query that selects pregnant women with high values of temperature and ECG in SETIF city this week. The Semantic Request Analyzer Agent automatically decomposes the query into structured sub-requests: Location constraint: SETIF, Time constraint: current week, Health conditions: high temperature, abnormal ECG, QoS requirement: low latency. These elements are mapped to ontology concepts using SAREF4EHAW and S-CQRS ontologies.

Step 2. Semantic Discovery and Filtering. After the semantic request analyzing and decomposing, the semantic microservice discovery and filtering agent discovers and filters the relevant health microservices to fulfill both user's sub-requests and QoS preferences. This filtering is important for deploying only needed sensors-based microservices. The discovery process is based on SAREF4EHAW ontology combined with S-CQRS ontology, semantic microservices description registry, and health thesaurus. First, the semantic microservice discovery and matching agent computes the similarity between the type of request (e.g., health information and monitoring, analysis, treatment) and the category of microservice. Once all similarities $\mathrm{Sim}_1\ (CT_{\mathrm{U}}, CT_{\mathrm{MS}})$ are calculated, the

similarity between pairs of concepts of both semantic microservices and user's sub-requests is computed according to two similarity (Sim_1, Sim_2) .

Then, the comparison will be performed between classified microservices and user's sub-requests in terms of type of data (e.g., scalar, multimedia), category of data (e.g., temperature, ECG, GPS), and disease family (e.g., gynecology, diabetes).

Finally, the resulting relevant microservices are ranked by user QoS preferences and returned to the Semantic Microservice CQRS Broker. Table 3 shows a list of the user's requests and related concepts, Table 4 shows examples of microservices classified by category, and Table 5 shows a list of relevant microservices for pregnancy disease.

In our cqse study, The Discovery and Matching Agent retrieves candidate microservices from the registry and performs multi-level filtering, functional matching (temperature, ECG, localization), semantic similarity computation (Sim_1 $(CT_{\text{U}}, CT_{\text{MS}})$), context matching (location, time) and QoS filtering (latency constraints). The result is a ranked list of relevant microservices, such as Temperature Monitoring Service, ECG Monitoring Service, Localization Service, HighTemperature Detection Service and AbnormalECG Detection Service.

Table 3. Types de requêtes et concepts associés

Sub-requests	Type	Concepts
Sub-Req#1	Read	Patient, ECG, Temperature
Sub-Req#2	Read	Patient, ECG, Temperature, City, Date

Step 3. Automated Orchestration.. After the filtering process, the semantic microservice orchestrator agent will orchestrate a list of relevant microservices received from the previous step. It takes as inputs the relationships among the user's sub-requests. It matches also the inputs and outputs of microservices and then generates automatically the composition of microservices. The Semantic Microservice Orchestrator Agent automatically constructs the service chain based on semantic input/output dependencies, output of Temperature Monitoring to input of High Temperature Detection, output of ECG Monitoring to input of Abnormal ECG Detection, output of Localization Service combined with diagnostic services. Using ontology reasoning, the orchestrator ensures compatibility between service inputs and outputs, correct execution order and dependency resolution between services.

The resulting automatically generated workflow is: Localization → Temperature Monitoring → High-Temperature Detection → ECG Monitoring → Abnormal ECG Detection → Aggregation → Result Delivery.

Step 4. Execution and Adaptation. The Execution Agent deploys and executes the composed microservices using REST APIs.The system continuously monitors context changes and adapts automatically..

The most important task of the semantic microservice executor agent comes from microservice command/query agent which generates the orders interpreted by the DS-CQRS platform to deploy or stop appropriate write/read microservices on IoT-based Pregnancy Women Monitoring Application. In Figure 13, we present only the microservices required to respond to the physic's request. The Execution microservice agent uses REST API to launch Localization, temperature, and ECG microservices to read sensed values, save position GPS, and synchronize patient data from the external system. When "HighTemperature" microservice detects that the temperature value is very high (e.g., exceeds 39) then "Localization" microservice sends patient's current location to a physics and "Prescribe_Medic" provided appropriate medications accordingly.

A second scenario is when the platform detects the temperature sensor's failure. A notification is sent to the Semantic discovery and matching agent, and the temperature microservices that measured temperature in °C is dynamically updated by another equivalent microservice that measures temperature in Kelvin. The platform adapts the application by stopping the current temperature microservice and deploying a new microservice that ensures timely data for continuous health monitoring.

A third scenario is when the application detects high-temperature level which can be a possible risk for diabetes disease. For this case, the physics recommends measurement of glucose level and changes his request to get the current glucose level of a pregnant woman and a ChangeRequestEvent event is generated. A notification is sent to the Semantic discovery and matching of microservice, and the list of relevant microservices is

Table 4. List of relevant microservices based on request of physics

Service Role	Service Type	Service Name	Service Concepts
Monitoring and Health Information	Read	Get_Profile	Patient's profil
		Get_Patient_Status	Patient's status
		Get_Localization	Location, Network
		Get_Temperature	DateTime, Patient, Temperature
		Get_ECG	DateTime, Patient, ECG
		Get_BloodPressure	DateTime, Patient, Blood Pressure
		Get_Weight	DateTime, Patient, Weight
	Write	Measure_Temp_C	DateTime, Patient, Temperature
		Measure_Temp_K	DateTime, Patient, Temperature
		Measure_Weight	DateTime, Patient, Weight
Analysis	Read	Health_Data_Analyzer	DateTime, Patient, Health Data
		High_Temperature	DateTime, Patient, Temperature
	Write	Analyse_Report	DateTime, Patient, Report
		Risk_Of_Diabet	DateTime, Patient, Diabetes
Treatment	Read	Receive_Report	Patient, Health Data
		Deliver_Report	DateTime, Patient, Daisies
	Write	Prescribe_Medic	DateTime, Patient, Medication
Danger	Read	SMS_Notification	DateTime, SMS
		Email_Notification	DateTime, Email
		Emergency_Call	DateTime, Emergency's phone
		Familly_Call	DateTime, Family's phone

Table 5. Scores de similarité et rangs des microservices

Microservices	$\mathbf{Sim_{ms\text{-}req}}$	Rang
Health_Data_Analyzer	0.881	1
Receive_Report	0.881	1
Analyse_Report	0.881	1
Deliver_Report	0.808	2
Get_ECG	0.654	3
Get_Temperature	0.654	3
Measure_Temp_C	0.654	3
Measure_Temp_K	0.654	3
Get_Weight	0.618	4
Measure_Weight	0.618	4
Measure_ECG	0.618	4
Prescribe_Medication	0.574	5
Get_Localization	0.537	6
Get_Patient_Status	0.519	6
Get_Blood_Pressure	0.498	7
Measure_Blood_Pressure	0.498	7
Get_Profile	0.333	8
Define_Profile	0.333	8

dynamically updated as shown in Table 5. Of course, these microservices will be orchestrated. The platform deploys "Get_GlucoseLevel", "High_GlucoseLevel", "Emergency_Call", and "Adjust_Insulin" microservices which can

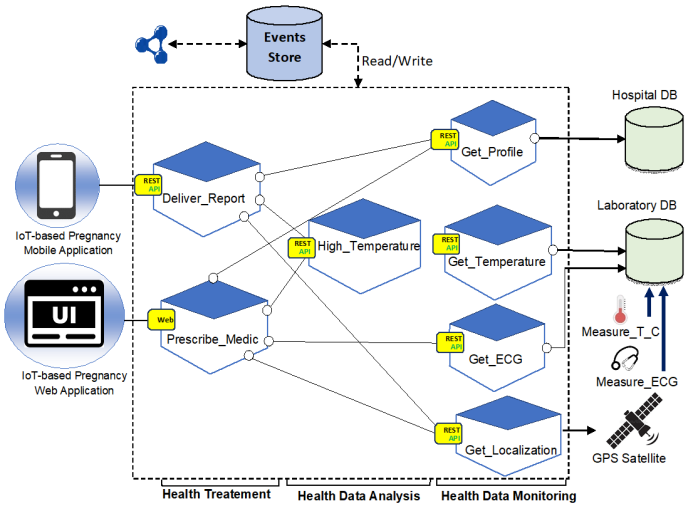


Figure 17. Deploying microservices in the application and receiving analysis results

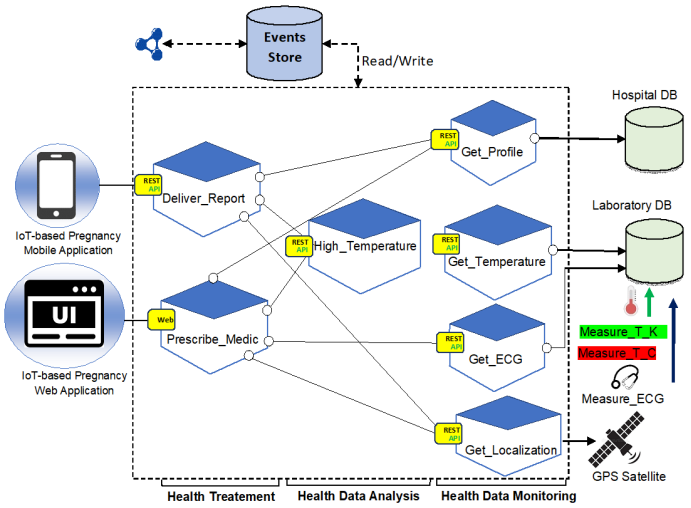


Figure 18. Deploying microservices in the application after the detection of temperature sensor's failure

find useful in a current health situation. However, when a high glucose level is detected by the "High_GlucoseLevel" microservice, a call to the local healthcare services is triggered based on the current position of the patient. Moreover, the application is capable of considering the health situation of the patient profile and can change insulin dosages appropriately.

5.6. Experimental Comparison

5.6.1. Evaluating the efficiency of DS-CQRS We evaluate the effectiveness of the proposed DS-CQRS framework in comparison with three baseline approaches, namely: (i) a standard CQRS-based microservices architecture without semantic enhancement, (ii) the SAREF4EHAW ontology-based semantic discovery approach, and (iii) the semantic microservices description and discovery method proposed by Surianarayanan et al. [27]. The evaluation is conducted on a set of dynamically generated healthcare service requests as described in Section 5.2.

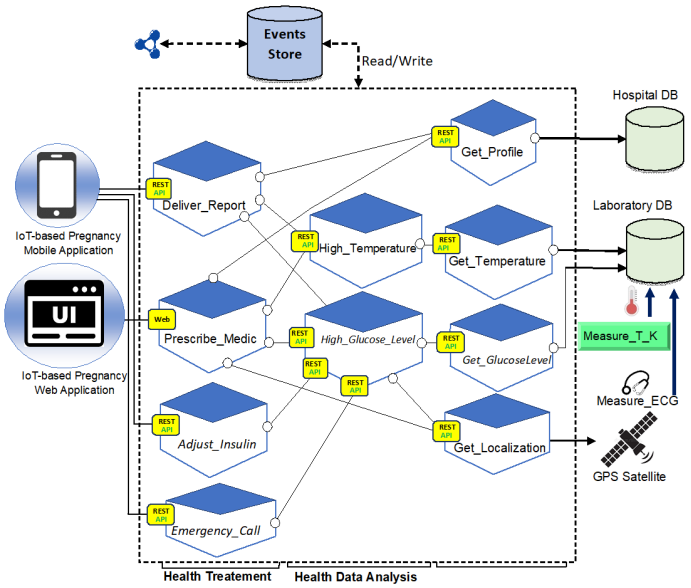


Figure 19. Deploying microservices in the application after detection risk of diabetes

Figure 20, Figure 21 show the performance accuracy and recall, respectively, with a different number of microservices and different kinds of daises for all approaches. Overall, DS-CQRS consistently outperforms the baselines, achieving accuracy between 0.88 and 0.93, recall between 0.85 and 0.91, and F2-scores up to 0.90. In contrast, the CQRS-only approach shows lower performance (accuracy: 0.75–0.80) due to the lack of semantic reasoning.

The SAREF4EHAW-based approach improves semantic interoperability (accuracy: 0.82–0.86) but remains limited by its static nature. Similarly, the method proposed by Surianarayanan et al. [27] achieves moderate performance (accuracy: 0.83–0.87) but struggles to handle dynamic context changes.

The CQRS-only model benefits from efficient query processing but lacks semantic understanding, resulting in lower recall and F2-score. The SAREF4EHAW approach enhances interoperability but cannot adapt to real-time context variations, leading to moderate performance. The approach by Surianarayanan et al. [27] provides effective semantic discovery but lacks dynamic orchestration capabilities, limiting its adaptability in changing environments.

In contrast, DS-CQRS integrates semantic reasoning, context-awareness, and CQRS optimization, enabling both efficient data retrieval and adaptive service matching. The use of context similarity and QoS-aware filtering improves recall, while CQRS caching reduces response time. This balanced improvement is reflected in higher F2-scores, demonstrating the robustness of DS-CQRS in dynamic smart healthcare environments.

5.6.2. Evaluating the effectiveness of DS-CQRS Fig. 22 presents the response time comparison of the proposed DS-CQRS framework against three baseline approaches under varying numbers of microservices. The graph illustrates the scalability behavior of each approach as system complexity increases.

Although semantic processing introduces additional computational overhead compared to the CQRS-only approach, the overall response time of DS-CQRS remains lower than both SAREF4EHAW and Surianarayanan et al. [27] approaches due to efficient filtering and reduced search space.

From a scalability perspective, the performance of all approaches degrades as the number of microservices increases; however, DS-CQRS exhibits a slower growth rate in response time. This improvement is achieved by semantically filtering candidate services before applying matching algorithms, while CQRS caching accelerates repeated queries.

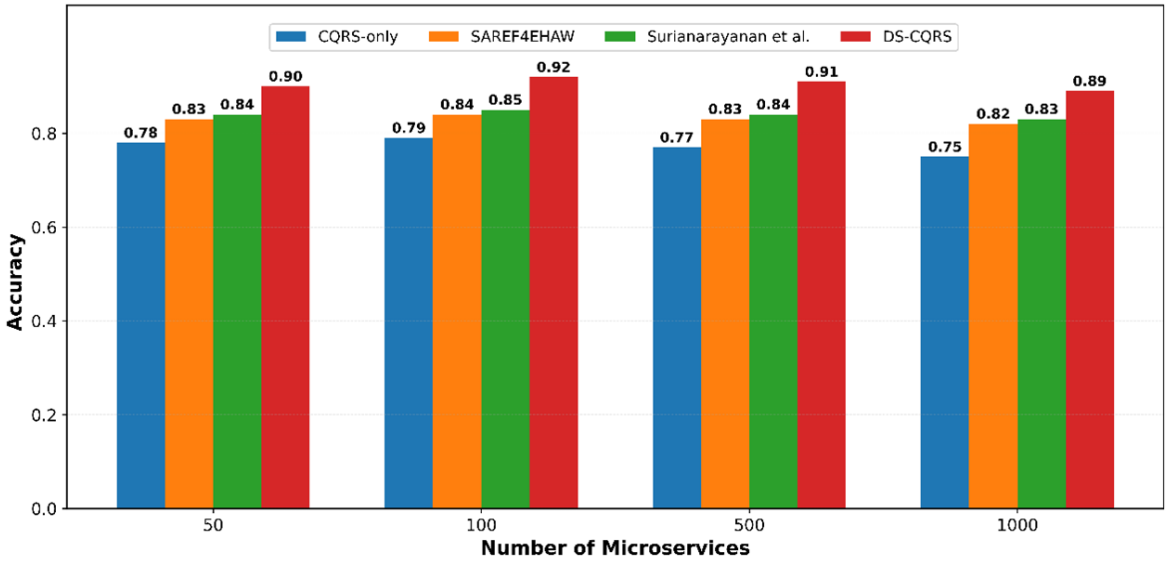


Figure 20. Accuracy comparison between the proposed DS-CQRS and other related works for different patient contexts

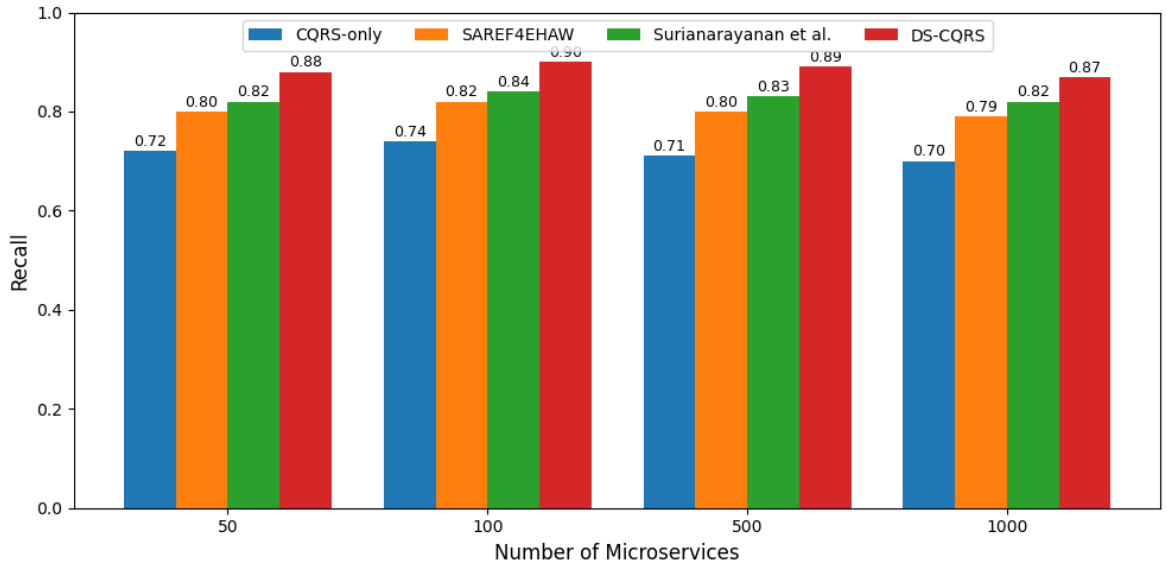


Figure 21. Recall comparison between the proposed DS-CQRS and other related works for different patient contexts

Overall, the experimental results demonstrate that DS-CQRS provides a balanced trade-off between semantic intelligence and system efficiency, outperforming existing approaches in terms of accuracy, recall, and response time, particularly in dynamic and context-rich smart healthcare environments.

5.6.3. Computational Complexity To provide a theoretical understanding of the scalability behavior of the proposed framework, we analyze the computational complexity of the service matching and retrieval processes with respect to the number of available microservices.

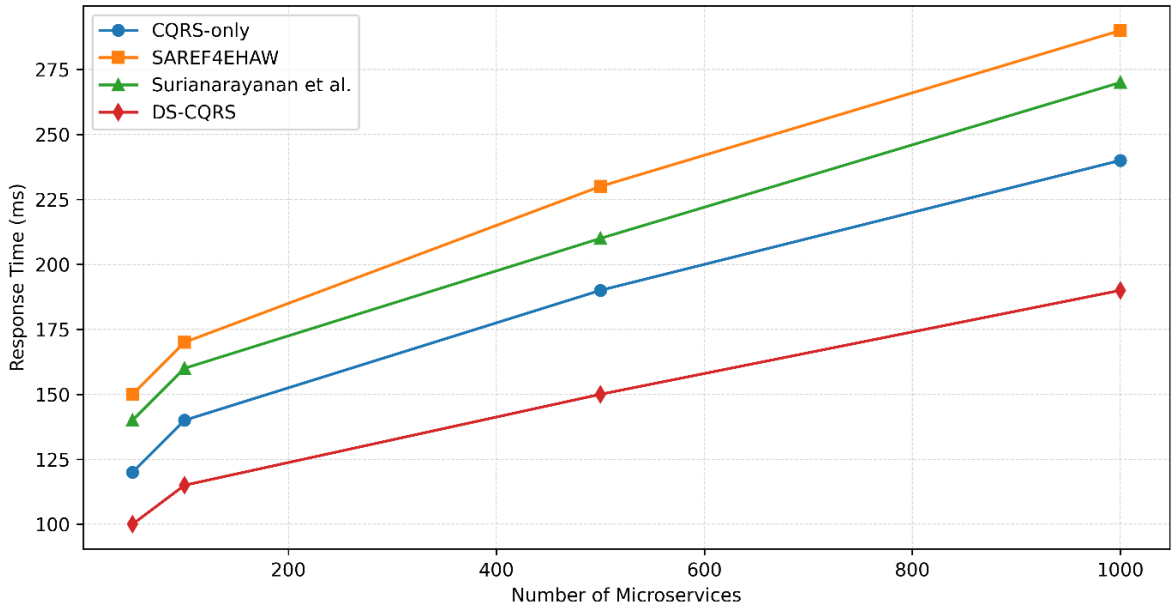


Figure 22. Response time comparison between the proposed DS-CQRS and other related works for different patient contexts

- Semantic Matching: $O(N \log N)$ (due to ranking and similarity computation)
- CQRS Retrieval: $O(1)$ (constant time for cached queries)

Let N be the number of microservices:

- CQRS-only: $O(N)$
- SAREF4EHAW: $O(N \times C)$, where C is the ontology reasoning cost
- Surianarayanan et al.: $O(N \times S)$, where S is the semantic similarity computation cost
- DS-CQRS: $O(K \times S)$, where $K \ll N$ due to context-aware filtering

Unlike baseline approaches that perform similarity evaluation over the entire service set, the proposed DS-CQRS framework first reduces the search space by applying context-aware and QoS-based filtering mechanisms. This significantly limits the number of candidate services that require semantic similarity computation. Consequently, by reducing K, the proposed approach effectively lowers the overall computational cost while maintaining high matching accuracy and recall. This reduction enables the framework to scale efficiently in large and dynamic environments, making it particularly suitable for real-time smart healthcare systems where both performance and responsiveness are critical.

Moreover, the DS-CQRS architecture introduces a read model caching mechanism, enabling constant-time retrieval for repeated or similar queries. This optimization reduces redundant computations and contributes to improved system responsiveness.

6. Discussion

6.1. Findings

Context is key for most healthcare decisions across patients. In today's competitive healthcare delivery, many contextual factors influence healthcare decision, making it costly, dynamic, and difficult. However, most healthcare

services still use simple methods supplemented by manual service adaptation. Thus, heartcare organizations need to invest a considerable amount of effort into semantic approaches and smart technologies in the presence of emergency and sensitive healthcare situations. Our re-search suggests that using CQRS pattern, IoT and context-aware semantic microservices can help automate and guide the distributed healthcare service delivery process. This will save both safety and time for patients. They can use the time saved for other heartcare operational tasks Furthermore, ontology models coupled with a CQRS pattern can improve the quality of the decision-making process with (0.956) accuracy. On the other hand, DS-CQRS approach has marked minimal execution time as compared to WordNet approach.

6.2. Limitations

Clearly, our study is limited to the domain of our case study, which comprises data from pregnant women for three disease categories (Anemia, Diabetic, COVID-19). Thus, it may not be generalizable to other disease categories without appropriate customiza-tion. Therefore, how to incorporate other contextual factors such as multiple disease types, emergency events, and other health crises might be an interesting future research avenue. Additionally, we did not consider emergencies with different priority levels. Thus, leveraging an adaptive emergency strategy with an optimal waiting time and respect to priority information is a potential avenue for future work.

The proposed CxS-CQRSontology is not restricted to a specific healthcare scenario (e.g., pregnancy monitoring). However, it is designed as a modular, extensible semantic layer that can be adapted to multiple domains. In particular, the ontology can be extended in two main ways: (1) The S-CQRS ontology can be linked with well-established domain ontologies such as disease ontologies (e.g., general medical knowledge bases), IoT-specific ontologies like smart agriculture, and industrial ontologies, (2) The ontology can be expanded to include a full spectrum of healthcare concepts, such as diagnostic procedures, treatment protocols and patient history and risk factors.

Future work will extend the framework to multi-patient triage scenarios and emergency prioritization systems, enabling dynamic resource allocation under critical conditions. This will allow the system to dynamically allocate medical services and optimize response under high-demand conditions.

7. Conclusion

We have developed a new dynamic and semantic DS-CQRS approach in the healthcare field. The approach is based on context-aware semantic CxS-CQRS ontology combined with microservices and CQRS architectural pattern. The approach was evaluated by using different types of requests to select relevant microservices that help ensure highly accurate results. We also compared the proposed approach with the most popular approach WordNet. The evaluation of the obtained results was achieved through well-known metrics such as performance accuracy reduction and execution time. The results obtained are very encouraging and validate the efficiency of our approach. The proposed approach suffers from non-convergence of the recognition process for some datasets for both QDA and MLP classifiers. In future research, we will test the proposed approach on other fields and adapt it to multi-criteria dynamic optimization problems.

REFERENCES

1. X. Chen, H. Xie, Z. Li, G. Cheng, M. Leng, and F. L. Wang, *Information fusion and artificial intelligence for smart healthcare: a bibliometric study*, *Information Processing & Management*, vol. 60, pp. 1–14, 2023.
2. C. Du, *Logistics and Warehousing Intelligent Management and Optimization Based on Radio Frequency Identification Technology*, *Journal of Sensors*, pp. 1–11, 2021.
3. S. Aydin and M. N. Aydin, *Design and implementation of a smart beehive and its monitoring system using microservices in the context of IoT and open data*, *Computers and Electronics in Agriculture*, vol. 196, p. 106897, 2022.
4. G. Saravanan, S. S. Parkhe, C. M. Thakar, V. V. Kulkarni, H. G. Mishra, and G. Gulothungan, *Implementation of IoT in production and manufacturing: An Industry 4.0 approach*, *Materials Today: Proceedings*, vol. 51, pp. 2427–2430, 2022.
5. A. P. Ramallo-González, A. González-Vidal, and A. F. Skarmeta, *CIoTVID: Towards an Open IoT-Platform for Infective Pandemic Diseases such as COVID-19*, *Sensors*, vol. 21, p. 484, 2022.

6. C. K. Rath, A. K. Mandal, and A. Sarkar, *Microservice based scalable IoT architecture for device interoperability*, *Computer Standards & Interfaces*, vol. 84, p. 103697, 2023.
7. P. Malo-Perisé and J. Merseguer, *The "Socialized Architecture": A Software Engineering Approach for a New Cloud*, *Sustainability*, vol. 14, p. 2020, 2022.
8. R. Chandra, S. Agarwal, and N. Singh, *Semantic sensor network ontology-based decision support system for forest fire management*, *Ecological Informatics*, vol. 72, p. 101821, 2022.
9. A. Gyrard and A. Kung, *SAREF4EHAW-compliant knowledge discovery and reasoning for IoT-based preventive health and well-being: IoT-based preventive health and well-being knowledge discovery and reasoning*, *Semantic Models in IoT and Ehealth Applications*, vol. 179, pp. 171–198, 2022.
10. P. Mangwani, N. Mangwani, and S. Motwani, *Evaluation of a Multitenant SaaS Using Monolithic and Microservice Architectures*, *SN Computer Science*, vol. 4, p. 185, 2023.
11. S. Sadeghiram, H. Ma, and G. Chen, *Multi-objective distributed Web service composition—A link-dominance driven evolutionary approach*, *Future Generation Computer Systems*, 2023.
12. M. Söylemez, B. Tekinerdogan, and A. Kolukısa Tarhan, *Feature-Driven Characterization of Microservice Architectures: A Survey of the State of the Practice*, *Applied Sciences*, vol. 12, p. 4424, 2022.
13. A. Benayache, A. Bilami, S. Barkat, P. Lorenz, and H. Taleb, *MsM: A microservice middleware for smart WSN-based IoT application*, *Journal of Network and Computer Applications*, vol. 144, pp. 138–154, 2019.
14. A. Boulmakoul, L. Karim, and B. Bhushan, *Smart Trajectories: Metamodeling, Reactive Architecture for Analytics, and Smart Applications*, CRC Press, 2022.
15. A. Malki, *Middleware and Services, Chapter 5: Event Driven Micro-Services, CQRS, Event Sourcing, SAGA, Axon, Kafka*, *Middleware and Services*, Higher School in Computer Science, Sidi Bel Abbes, Algeria, 2020.
16. R. K. Shinde, M. S. Alam, S. G. Park, S. M. Park, and N. Kim, *Intelligent IoT (IIoT) Device to Identifying Suspected COVID-19 Infections Using Sensor Fusion Algorithm and Real-Time Mask Detection Based on the Enhanced MobileNetV2 Model*, *Healthcare*, vol. 10, p. 454, 2022.
17. X. Li, Y. Lu, X. Fu, and Y. Qi, *Building the Internet of Things platform for smart maternal healthcare services with wearable devices and cloud computing*, *Future Generation Computer Systems*, vol. 118, pp. 282–296, 2021.
18. M. S. Munir, I. S. Bajwa, A. Ashraf, W. Anwar, and R. Rashid, *Intelligent and smart irrigation system using edge computing and IoT*, *Complexity*, 2021.
19. M. G. Khan, N. U. Huda, and U. K. U. Zaman, *Smart Warehouse Management System: Architecture, Real-Time Implementation and Prototype Design*, *Machines*, vol. 10, p. 150, 2022.
20. A. M. Panchea, D. Létourneau, S. Brière, M. Hamel, M. A. Maheux, C. Godin, and F. Michaud, *OpenTera: A microservice architecture solution for rapid prototyping of robotic solutions to COVID-19 challenges in care facilities*, *Health and Technology*, vol. 12, pp. 583–596, 2022.
21. M. Taneja, N. Jalodia, J. Byabazaire, A. Davy, and C. Olariu, *SmartHerd management: A microservices-based fog computing–assisted IoT platform towards data-driven smart dairy farming*, *Software: Practice and Experience*, vol. 49, pp. 1055–1078, 2019.
22. X. Xu, Z. Chai, Z. Xiong, and J. Wu, *A Scalable Resource Management Architecture for Industrial Fog Robots*, *Proceedings of the International Conference on Intelligent Robotics and Applications*, pp. 67–77, 2021.
23. G. Ortiz, J. Boubeta-Puig, J. Criado, D. Corral-Plaza, A. Garcia-de-Prado, I. Medina-Bulo, and L. Iribarne, *A microservice architecture for real-time IoT data processing: A reusable Web of things approach for smart ports*, *Computer Standards & Interfaces*, vol. 81, p. 103604, 2022.
24. A. Mavrogiorgou, S. Kleftakis, K. Mavrogiorgos, N. Zafeiropoulos, A. Menychtas, A. Kiourtis, and D. Kyriazis, *beHEALTHIER: A microservices platform for analyzing and exploiting healthcare data*, *Proceedings of the 2021 IEEE 34th International Symposium on Computer-Based Medical Systems (CBMS)*, pp. 283–288, 2021.
25. D. S. S. Venkatesh and S. Agarwal, *Data Access Pattern Recommendations for Microservices Architecture*, *Proceedings of the 2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 241–243, 2022.
26. Z. Long, *Improvement and implementation of a high performance CQRS architecture*, *Proceedings of the 2017 International Conference on Robots & Intelligent System (ICRIS)*, pp. 170–173, 2017.
27. C. Surianarayanan, G. Ganapathy, and P. R. Chelliah, *A Novel Approach for Semantic Microservices Description and Discovery Toward Smarter Applications*, *Machine Intelligence and Smart Systems*, pp. 89–101, 2022.
28. E. Ntentos, U. Zdun, K. Plakidas, D. Schall, F. Li, and S. Meixner, *Supporting architectural decision making on data management in microservice architectures*, *Proceedings of the European Conference on Software Architecture*, pp. 20–36, 2019.
29. A. Alti, A. Lakehal, S. Laborie, and P. Roose, *Autonomic semantic-based context-aware platform for mobile applications in pervasive environments*, *Future Internet*, vol. 8, p. 48, 2016.
30. J. M. N. Ramírez, P. Roose, M. Dalmau, and Y. Cardinale, *An event detection framework for the representation of the AGGIR variables*, *Proceedings of the 2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 153–160, 2018.
31. M. Sanaa and L. Fadwa, *Towards a Context Awareness System Using IoT, AI, and Big Data Technologies*, *IEEE Access*, vol. 13, 2025.
32. N. Phu, A. N. Huu-H, P. Phu, T. Hong-Linh, and C. Thomas, *Advanced Context-Sensitive Access Management for Edge-Driven IoT Data Sharing as a Service*, *ACM Transactions on Internet Technology*, vol. 25, 2025.
33. Oluwaseun Bamgboye, Xiaodong Liu, Peter Cruickshank, and Qi Liu, *Semantic-Driven Approach for Validation of IoT Streaming Data in Trustable Smart City Decision-Making and Monitoring Systems*, *Big Data and Cognitive Computing*, vol. 9, 2025.
34. K. Hyojung, F. Lloyd, and P. John, *Analyzing temporal patterns in frequent emergency department visits among oncology patients using semantic similarity measures*, *American Journal of Emergency Medicine*, vol. 89, pp. 51–56, 2025.
35. B. Annane, A. Alti, and A. Lakehal, *Blockchain based context-aware CP-ABE schema for Internet of Medical Things security*, *Array*, vol. 14, p. 100150, 2022.