A Model for Refactoring Monolithic Applications to Microservices Using Domain-Driven Design: A Case Study on IoT Irrigation Systems

Munezero Immaculee Joselyne[1], Ngenzi Alexander[2], Hitimana Eric[3], Ipinnimo Oluwafemi [4]
School of ICT, Computer and Software Engineering, University of Rwanda, Kigali 3900, Rwanda[1,2,3]
Faculty of Engineering, Electrical and electronics, University of Lagos, Lagos, Nigeria[4]

***Abstract*—The increasing complexity of Internet of Things (IoT) applications has exposed the limitations of monolithic software architectures in addressing scalability, flexibility, and real-time processing requirements. Although microservice architectures offer a promising alternative, identifying optimal service boundaries remains a significant challenge, often resulting in excessive inter-service communication and degraded system performance when poorly defined. This study proposes a quantitative model for refactoring monolithic applications into microservices by integrating Domain-Driven Design (DDD) principles with measurable metrics, including service size, coupling, and scalability. The model systematically identifies optimal service boundaries through a structured evaluation framework. The proposed approach is validated using a case study of an IoT-based irrigation management system. Experimental results show a reduction in inter-service communication overhead and improved modularity and scalability compared to baseline decomposition approaches. The findings demonstrate that combining DDD concepts with quantitative analysis provides an effective and practical solution for guiding microservice migration in complex IoT environments. The average coupling score across the refactored system was recorded at 20.4%, which satisfies the theoretical requirement of remaining below 30% and aligns with empirical observations from successful microservice decompositions.**

***Keywords*—Microservice; refactoring model; IoT irrigation system; DDD**

I. Introduction

The rapid adoption of Microservice Architecture across the software industry has necessitated robust strategies for migrating legacy monolithic systems to more modular and scalable structures [1] [2]. While microservices offer advantages in terms of independent deployment and scalability, the transition from a monolithic structure presents significant challenges regarding service granularity, cohesion, and coupling that require systematic approaches to identify appropriate service boundaries [3] [4]. While Domain-Driven Design offers a valuable conceptual framework for identifying bounded contexts [5], the approach lacks concrete quantitative mechanisms to resolve the ambiguity in defining service boundaries, which often leads to suboptimal decompositions that fail to balance network latency, data consistency, and dependency management effectively [6]

To address this limitation, this study proposes a mathematical model that quantifies and formalizes architectural decision-making by integrating key metrics such as service size, scalability, and coupling, thereby establishing a formal basis for the decomposition of monolithic IoT systems into microservices [7]. This model provides a structured mechanism to evaluate maintainability by explicitly quantifying factors such as architectural complexity and cohesion, which are essential for long-term system sustainability.

The modernization strategies employing well-established methods such as the analysis of Domain-Driven Design patterns to locate potential microservices are essential for decomposing legacy code bases while managing the complexity inherent in functional redesign and organizational criteria [7][8]. The process of breaking down an application domain into properly sized components remains a challenging task because the notion of the ideal size for these components is often blurred or unclear, necessitating the integration of quantitative models to identify optimal service boundaries that balance network latency, data consistency, and dependency management [9]. Finding the optimum size for a service is particularly complex as it requires finding a balance that accommodates varying development costs, performance overheads, and the distinct requirements of different application domains [8], while balancing distinct factors like afferent and efferent coupling against relational cohesion to ensure the modularity and maintainability of the refactored architecture [7].

Therefore, this study introduces a mathematical framework that operationalizes DDD concepts by quantitatively evaluating service candidates against metrics of size, scalability, and coupling to reduce the ambiguity inherent in defining bounded contexts. This ambiguity arises due to the fact that because a bounded context is primarily a logical distinction denoting where specific domain rules apply, rather than a structural or physical boundary, which complicates the direct translation of domain models into deployable microservices without quantitative validation [10] . The clear quantitative measures for defining optimal boundaries and functional divisions are currently lacking [11], [9], [12].

The absence of standardized quantitative metrics for evaluating architectural trade-offs often forces architects to rely on intuition, increasing the risk of creating distributed systems that suffer from either excessive latency due to chatty inter-service communication or diminished autonomy due to overly coarse-grained service boundaries [13][1][4]. This lack of rigor can result in a distributed monolith, a system that suffers from the operational disadvantages of distributed architectures without gaining the intended modularity or independent scalability,

effectively trapping organizations in a state of diminished performance and increased complexity. To bridge this gap, existing literature has begun formalizing extraction models to support algorithmic recommendations, recognizing that while informal migration patterns exist, there is a fundamental lack of formal models and automated support tools to rigorously guide the decomposition process [7].

This research aims to develop and validate a quantitative model for refactoring monolithic applications into microservices using DDD principles. Specifically, the study seeks to: (1) formulate measurable metrics, including service size, coupling, and scalability, to support the systematic identification of optimal microservice boundaries; (2) evaluate the effectiveness of the proposed model in reducing inter-service coupling and improving system scalability compared to traditional decomposition approaches; (3) investigate the extent to which the model enhances modularity and architectural stability; and (4) assess the impact of the proposed approach on system performance and development efficiency through a case study of an IoT-based irrigation system.

The scalability validation showed that the sensor ingestion service handled a 50% increase in throughput without proportional latency growth, confirming the model's effectiveness in predicting workload scaling behavior in dynamic IoT environments.

II. Literature Review

Existing literature on decomposing large systems into microservices predominantly frames the refactoring task as a graph clustering problem, where static code analysis uncovers functional dependencies to inform service boundaries [14], [15]. Approaches using semantic coupling strategies couple classes based on shared domain model entities identified via extracting meaning from source code, aligning microservice boundaries with bounded contexts from Domain-Driven Design [12][16]. Model-based methodologies further extend UML component diagrams with formal profiles that incorporating Domain-Driven Design's bounded context patterns, enabling precise specification of microservice communications and subsequent transformations between models for code generation [8]. These transformations facilitate the automatic derivation of microservice architectures by mapping clustered domain entities into independent deployment units, preserving high cohesion within services while minimizing inter-service dependencies [17], [6]. Nevertheless, these methodologies often lack strict mathematical definition for optimizing cluster modularity, prompting the need for graph-theoretic models that quantify bounded context cohesion through density metrics on entity interdependence [15].

This study advances the field by introducing a mathematical model employing Domain-Driven Design's bounded contexts as optimization objectives within clustering algorithms, maximizing service cohesion through weighted edges representing domain entity couplings [8], [14].

Current microservice extraction research reveals diverse analytical approaches, including tracing runtime behavior combined with bounded context patterns to dynamically visualize service boundaries, functionality-oriented clustering of execution traces validated by cohesion and coupling metrics, and systematic analysis of functions, interfaces, dataflows, and storage [7][18]. These paradigms frequently encounter limitations in achieving domain-driven decompositions, as optimization models minimizing inter-service coupling via weights on connections fail to constrain entity placements within semantically meaningful bounded contexts [9]. The proposed model addresses these shortcomings by formulating bounded context identification as a combinatorial optimization over graph communities derived from domain entity couplings, yielding microservices with minimized inter-service dependencies, while preserving semantic cohesion[19]. Empirical validation of this model on legacy monolithic systems demonstrates its capacity to produce microservices adhering to the single responsibility principle through combinatorial optimization over high-cohesion graph communities[15]. Additionally, empirical assessments confirm the model's superiority through higher cohesion scores and reduced coupling metrics relative to baseline decompositions, and precision-recall alignments exceeding those of interface analysis and graph clustering benchmarks [6][20]. To put this model into operation, the subsequent sections delineate its mathematical formulation, commencing with graph construction from domain entities and proceeding to optimization objectives maximizing modularity via attention-augmented clustering [6].

A. Research Gap and Novel Contributions

Although existing microservice decomposition approaches have achieved promising results through graph clustering, semantic analysis, and optimization techniques[7], [9], several limitations remain. First, most graph-based approaches focus primarily on structural dependencies and community detection without explicitly incorporating scalability requirements into the decomposition process[21]. Second, optimization-based methods generally prioritize coupling minimization and cohesion maximization but do not simultaneously consider service size constraints[22], runtime workload distribution, and scalability objectives within a unified formulation.

The novelty of the proposed model lies in the integration of three complementary dimensions of software quality into a single optimization framework:

- Structural Quality through service size constraints that prevent both oversized and excessively fragmented microservices.
- Runtime Performance through scalability metrics derived from interaction frequencies and workload distributions.
- Architectural Modularity through coupling minimization and cohesion maximization guided by Domain-Driven Design bounded contexts.

Unlike conventional graph clustering approaches that generate partitions solely from dependency structures [23],[16], the proposed model incorporates bounded-context knowledge from Domain-Driven Design and evaluates candidate decompositions using a multi-objective optimization function. This enables the identification of service boundaries that simultaneously satisfy modularity, scalability, and maintainability requirements.

Furthermore, while existing optimization approaches typically rely on static code metrics [24], [25], the proposed framework combines static analysis, runtime interaction analysis, and co-change evolutionary coupling information to construct a richer representation of service dependencies. Consequently, the model provides a more comprehensive basis for microservice extraction in large-scale IoT applications.

III. Research Methodology

This study adopts a Design Science Research (DSR) methodology to develop and evaluate a mathematical model for refactoring monolithic applications into microservices. The DSR approach is appropriate as it focuses on the creation and validation of an artifact designed to address a specific engineering problem. In this context, the artifact is a quantitative model that supports the identification of optimal microservice boundaries based on measurable architectural attributes. The research process follows established DSR stages. First, the problem identification and motivation are established through a systematic literature review (SLR), which highlights the limitations of existing approaches for microservice decomposition, particularly in defining appropriate service boundaries. Second, the objectives of the solution are defined, focusing on the development of a systematic and quantitative method for designing scalable and modular microservice architectures. Third, the design and development phase involves constructing a mathematical model that integrates key metrics, including service size, coupling, and scalability. These architectural attributes are formalized into mathematical functions to enable the systematic evaluation of decomposition alternatives, with the aim of maximizing cohesion and scalability while minimizing inter-service coupling. Fourth, the proposed model is demonstrated through its application to a case study of an IoT-based irrigation system. Finally, the evaluation is conducted using a quantitative case study approach, where the performance of the proposed model is compared against traditional decomposition methods based on defined metrics. The results are then communicated to the research community to validate the effectiveness and applicability of the proposed approach.

IV. Development of the Refactoring Mathematical Model

A. Metrics for Service Size, Scalability, and Service Coupling

1) Service size: Determining the appropriate granularity for microservices is a complex challenge, as size is not a reliable standalone indicator for optimal decomposition, making it necessary to prioritize cohesion as the primary guideline rather than mere granularity [8][26]. This equilibrium is critical because research indicates that identifying the optimal granularity requires balancing various aspects, where simply focusing on low coupling and high cohesion values may not be sufficient for optimal modularity [5]. To quantify this balance objectively, specific metrics such as the number of methods and lines of code are utilized to measure service size, ensuring that the service scope remains within the recommended 10-100 lines of code range to facilitate efficient testing, deployment, and maintenance processes. To calculate the service size, this research utilizes a simple cardinality-based metric defined as [see Eq. (1)]:

$$S_i = |m_i| \tag{1}$$

where, S_i represents the size of microservice i, and m_i denotes the set of classes (or components) contained within the microservice. The optimal granularity of a microservice directly affects the application's quality attributes and the utilization of computational resources. These metrics evaluate the ability of an individual service to operate independently. This independence can be quantified by the ratio of external dependencies to total operations, where a higher value indicates that the service can be developed, deployed, and maintained with minimal impact on other system components [27].

2) Scalability potential: The scalability potential of a microservice is defined as a function of its size and internal structure [see Eq. (2)]:

$$V_i = \frac{H_i}{1 + S_i} \tag{2}$$

where, V_i denotes the scalability potential of service i, S_i is the service size, and H_i represents the cohesion within the service. This formulation allows efficient resource allocation and dynamic scaling, as services can be independently replicated based on demand. However, independent scaling must be balanced against interdependence introduced by service size.

3) Cohesion within microservices: Cohesion reflects the degree of relatedness among components within a microservice and is defined as [see Eq. (3)]:

$$H_i = \frac{1}{|m_i|^2} \sum_{c_p, c_q \in m_i} \text{sim}(c_p, c_q) \tag{3}$$

where, $\text{sim}(c_p, c_q)$ denotes the similarity between classes c_p and c_q within the same microservice.

4) Coupling between microservices: Coupling represents the degree of interdependence between distinct services. It is defined as [see Eq. (4)]:

$$C = \sum_{i \neq j} \sum_{c_p \in m_i} \sum_{c_q \in m_j} d(c_p, c_q) \tag{4}$$

where, $d(c_p, c_q)$ represents the dependency strength between class c_p in microservice m_i and class c_q in microservice m_j.

To rigorously evaluate interdependencies, we divided coupling into:

- Afferent Coupling (C_a): Measures incoming dependencies, defined as the number of external classes depending on classes within a microservice:

$$C_a(i) = |\{c \notin m_i \mid c \to m_i\}|$$

- Efferent Coupling (C_e): Measures outgoing dependencies, defined as the number of external classes that a microservice depends on:

$$C_e(i) = |\{c \notin m_i \mid m_i \to c\}|$$

Coupling is analyzed in both internal and external dimensions to evaluate its impact on maintainability and independent evolution. This ensures that the model captures both intra-service complexity and inter-service connectivity within the distributed architecture.

B. Unified Model for Service Decomposition

Fig. 1 presents the unified model procedure.

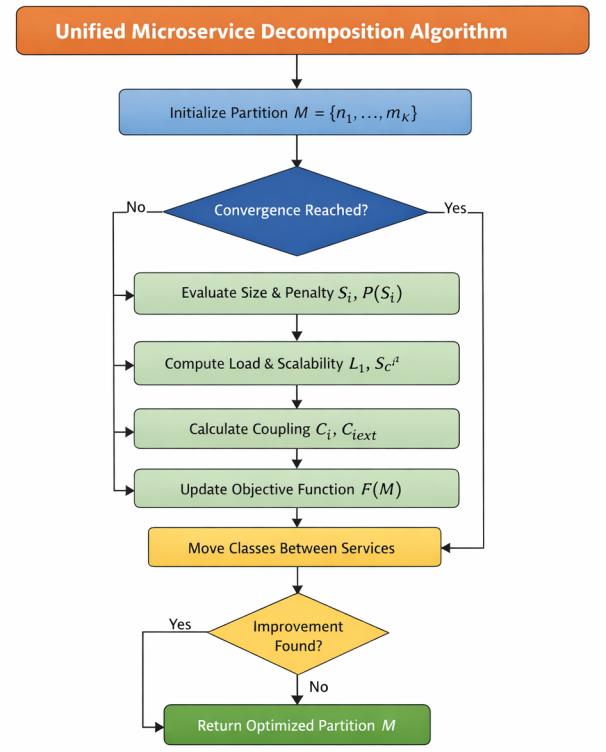

Fig. 1. Unified model procedure

Let the monolithic application be represented as a set of classes:

$$C_M = \{c_1, c_2, \ldots, c_N\}$$

which are partitioned into K disjoint microservices:

$$M = \{m_1, m_2, \ldots, m_K\}$$

such that:

$$\bigcup_{i=1}^{K} m_i = C_M, \qquad m_i \cap m_j = \emptyset \quad \forall i \neq j$$

The structural complexity of each candidate microservice is quantified using a composite size metric that aggregates cyclomatic complexity and method count. Service Size S_i can be represented as [see Eq. (5)]:

$$S_i = \sum_{c \in C_i} (V(G)_c + M_c) \tag{5}$$

where, C_i = set of classes in service i, $V(G)_c$ = cyclomatic complexity of class c, M_c = number of methods in class c, α = weighting factor. This formulation ensures that each service maintains a manageable level of complexity while avoiding excessive aggregation of responsibilities.

To enforce balanced granularity, the following constraint is imposed:

$$S_{\min} \leq S_i \leq S_{\max}$$

We imposed the constraint on the exact size of the microservice to discourage the size violation as:

$$P(S_i) = \max(0, S_{\min} - S_i)^2 + \max(0, S_i - S_{\max})^2$$

Scalability is evaluated using runtime interaction and resource consumption metrics. Let $I(f_p, f_q)$ denote the invocation frequency between functions. The normalized load distribution is [see Eq. (6)]:

$$L_i = \frac{1}{T} \sum_{(f_p, f_q) \in E_i} I(f_p, f_q) \tag{6}$$

where, load model L_i can be presented as: F_i = set of functions in service i, $I(f_p, f_q)$ = invocation count between functions f_p and f_q, $E_i \subseteq F_i \times F$ = interactions involving service i, $T = \sum_{(f_p, f_q) \in E} I(f_p, f_q)$ = total system-wide invocations. This gives a normalized estimate of how much runtime activity is associated with service i.

Coupling metric is presented as:

$$C_i = \omega_{ec} C_{ei} + \omega_{ac} C_{ai}$$

where C_{ei} = efferent coupling (outgoing dependencies), C_{ai} = afferent coupling (incoming dependencies), $\omega_{ec}, \omega_{ac} \in [0, 1]$ = weighting coefficients.

Then the frequency weighted coupling is represented as follows for the set of outgoing dependency pairs:

$$C_{ei} = \frac{1}{T} \sum_{(f_p, f_q) \in D_i^{out}} I(f_p, f_q)$$

and the set of incoming dependency pairs being represented as [see Eq. (7)]:

$$C_{ai} = \frac{1}{T} \sum_{(f_p, f_q) \in D_i^{in}} I(f_p, f_q) \tag{7}$$

After representation of the different parameters, we partition the candidates of microservice by considering the constants of minimizing the size while minimizing the coupling and maximizing the scalability, which can be presented as [see Eq. (8)]:

$$M_i = \sum_i \left[\lambda_1 \frac{1}{S_i} + \lambda_2 C_i - \lambda_3 L_i \right] \tag{8}$$

C. Model Variable Parameters for Partitioning and Optimization

To operationalize the model, we defined the core decision variables as $V = \{c_1, \ldots, c_n\}$: set of classes follows, and $M = \{m_1, \ldots, m_k\}$: set of microservice the variables are assigned to $x_{i,j} \in \{0, 1\}$:

$$x_{i,j} = \begin{cases} 1 & \text{if class } c_i \text{ is assigned to } m_j \\ 0 & \text{otherwise} \end{cases}$$

Ensure that each class is exclusively mapped to a single service boundary during the partitioning process.

Additionally, to represent the inter-service connectivity between classes c_i and c_j belonging to different microservices, a binary edge variable $y_{ij} \in \{0, 1\}$:

$$y_{ij} = \begin{cases} 1 & \text{if } c_i, c_j \text{ are in the same service} \\ 0 & \text{otherwise} \end{cases}$$

This is linking to constraint formulation [see Eq. (9)]:

$$y_{ij} = \sum_{k=1}^{|M|} x_{i,k} \cdot x_{j,k} \tag{9}$$

where, it can allow the objective function to account for the cut weight and minimize coupling as was done in [7].

Consequently, the coupling objective is formulated as in [7], [8] by minimizing the sum of weights for edges crossing partition boundaries, where E is the set of edges and w_{ij} the coupling weight is expressed as [see Eq. (10)]:

$$\min_{(i,j) \in E} w_{ij}(1 - y_{ij}) \tag{10}$$

This coupling strength w_{ij} further incorporates contributor-based coupling metrics, where the weight is defined as the cardinality of the intersection of developer sets between connected classes c_i and c_j.

We incorporated this approach in calculating the weight of coupling by utilizing the co-change frequency observed in the version control history to establish a dynamic coupling metric, defined as:

$$\delta(i, j) = \begin{cases} 1 & \text{if } c_i, c_j \text{ changed together in commit } h \\ 0 & \text{otherwise} \end{cases}$$

as the aggregated logical coupling is delivered as:

$$\Delta_{ij} = \sum_{h \in H_m} \delta(i, j)$$

and the final coupling weight is [see Eq. (11)]:

$$w_{ij} = |Dev(c_i) \cap Dev(c_j)| + \Delta_{ij} \tag{11}$$

This logical coupling metric serves as a crucial indicator of evolutionary dependencies, ensuring that classes that frequently change together are grouped into the same microservice boundary to maintain high cohesion.

We balanced this internal cohesion with the constraints of service size. Our model imposes a constraint on the number of classes assigned to each microservice to prevent the creation of overly complex or trivial services, as it is used in [4]. The constraint of size over cohesion to promote the generation of evenly distributed and manageable services is formulated as [see Eq. (12)]:

$$|m_j| = \sum_{i \in V} x_{i,j}, \quad \text{where } L_{\min} \leq |m_j| \leq L_{\max}, \quad \forall j \tag{12}$$

Beyond these structural constraints, this model incorporates a service cohesion metric to quantitatively assess the degree to which elements within a candidate microservice belong together based on communicational and textual similarities. For a service or cluster:

E_j^{int} : internal edges and V_j : nodes in the cluster

$$Cohesion_j = \frac{|E_j^{int}|}{|V_j|}$$

with a global cohesion [see Eq. (13)]:

$$\max \sum_{j=1}^{|M|} \frac{|E_j^{int}|}{|V_j|} \tag{13}$$

A high cohesion value suggests that all the methods inside a microservice have strong relationships with entities included in the same microservice, which implies that the service is well concerned with a specific bounded context. Conversely, low cohesion indicates that a service encompasses disparate functionalities that lack strong interconnectivity, potentially negating the benefits of modularization.

To ensure the robustness of the proposed decomposition for this model, the aim is to employ a multi-objective evaluation criterion that simultaneously optimizes for minimized inter-service coupling, maximized intra-service cohesion, and adherence to defined service size constraints.

We completed this model by incorporating the modularity quality metric, which serves as a comprehensive indicator of

the partitioning effectiveness by calculating the ratio of internal interactions to total interactions, expressed as:

$$Q = \frac{W_{intra} - W_{inter}}{W_{total}}$$

where [see Eq. (14)],

$$W_{intra} = \sum_{(i,j)\in E} w_{ij} y_{ij}, \quad W_{inter} = \sum_{(i,j)\in E} w_{ij}(1 - y_{ij}),$$

$$W_{total} = \sum_{(i,j)\in E} w_{ij} \tag{14}$$

This metric evaluates the structural cohesiveness of classes within a partition against the coupling between partitions, where a higher ratio indicates superior partition quality by ensuring that intra connectivity significantly outweighs the inter-connectivity between identified microservices.

From all the parameters defined above, we delivered a multi-object optimization as:

$$\min_{(i,j)\in E} w_{ij}(1 - y_{ij}), \quad \max_{j} Cohesion_j$$

This is subject to [see Eq. (15)]:

$$L_{\min} \leq \sum_{i} x_{i,j} \leq L_{\max} \tag{15}$$

The unified objective function is derived as follows [see Eq. (16)]:

$$\max \left[\lambda_1 \frac{W_{intra} - W_{inter}}{W_{total}} \right.$$

$$\left. + \lambda_2 \sum_{j} \frac{|E_j^{int}|}{|V_j|} - \lambda_3 \sum_{(i,j)\in E} w_{ij}(1 - y_{ij}) \right] \tag{16}$$

This final model utilizes quality metrics such as coupling, cohesion, complexity, and the size employed to determine the optimal clustering output and maintainability quality, where higher scores are assigned to configurations demonstrating high cohesion and low coupling.

D. Optimal Microservice Partitioning Algorithm

The optimal microservice partitioning algorithm is presented in Algorithm 1.

Algorithm 1 Optimal Microservice Partitioning

Require: Class set $C_M = \{c_1, c_2, \ldots, c_N\}$, dependency graph $G(V, E)$
Require: Weight parameters $\alpha, \omega_{ec}, \omega_{ac}, \lambda_1, \lambda_2$
Require: Constraints $S_{\min}, S_{\max}, L_{\min}, L_{\max}$
Require: Static metrics: cyclomatic complexity $V(G)_c$, method counts MC_c
Require: Commit history for co-change extraction
Require: Initial empty partition matrix $X = [x_{i,j}]$
Require: Feasible initialization of service sizes
$X \leftarrow$ random feasible assignment of classes to microservices
$M^* \leftarrow \emptyset$
$t \leftarrow 0$
Compute initial coupling weights w_{ij}
while convergence criterion is not satisfied **do**
  $t \leftarrow t + 1$ ▷ Update iteration counter
  Compute structural metrics $S_i,\ V(G)_c,\ MC_c$
  Compute scalability metric L_i
  Compute cohesion for each service $Cohesion_j$
  Update coupling weights w_{ij}
  Evaluate modularity Q
  Evaluate objective function $F(M)$
  Update class assignments $x_{i,j}$ using optimization operator
  (Non dominated Sorting Genetic Algorithm II (NSGA-II)
  Enforce constraints:
  $S_{\min} \leq S_i \leq S_{\max}$
  $L_{\min} \leq |m_j| \leq L_{\max}$
  $\sum_{j=1}^{K} x_{i,j} = 1, \quad \forall c_i \in C_M$
  if $F(M) > F(M^*)$ **then**
    $M^* \leftarrow M$
    $F(M^*) \leftarrow F(M)$
  end if
end while

E. Optimization Configuration

The unified objective function was optimized using the Non-dominated Sorting Genetic Algorithm II (NSGA-II)[25], a widely adopted evolutionary algorithm for multi-objective optimization problems.

The optimization process employed the following parameter settings: Population size: 100 candidate decompositions, Number of generations: 500, Crossover probability: 0.85, Mutation probability: 0.10, Tournament size: 2 Elitism strategy: Enabled

Each candidate solution was encoded as a class-to-service assignment vector. The fitness function was computed according to Eq. (16), balancing modularity, cohesion, and coupling objectives. The optimization process terminated when either:

- The maximum number of generations was reached, or
- The improvement in objective value remained below 0.001 for 50 consecutive generations.

This convergence criterion ensured computational efficiency while maintaining solution quality.

F. Integration with Domain-Driven Design (DDD)

The model we presented incorporates Domain-Driven Design concepts, specifically the definition of bounded contexts, to ensure that the refactoring process aligns the technical boundaries of microservices with the distinct business domains they represent. By leveraging these DDD principles, the model resolves the inherent ambiguity in defining bounded context boundaries by supplementing logical domain divisions with quantitative coupling and cohesion metrics to ensure high modularity and optimized service granularity [8]. This proposed model complements Domain-Driven Design by transforming domain concepts into quantifiable optimization criteria.

Initially, DDD is used to define candidate bounded contexts; these candidates are then refined using the optimization algorithm, where each m_i corresponds to a hypothesized bounded context derived from domain analysis:

$$M^{(0)} = \{m_1, m_2, \ldots, m_K\}$$

G. Mapping Between DDD and Model

Table I presents the mapping between DDD concepts and mathematical model components.

TABLE I. MAPPING BETWEEN DDD CONCEPTS AND MATHEMATICAL MODEL COMPONENTS

DDD Concept	Mathematical Representation
Bounded Context	Microservice m_i
Domain Cohesion	Size metric S_i
Service Autonomy	Scalability Sc_i
Context Independence	Low Coupling C_i^{ext}

V. CASE STUDY: IOT IRRIGATION SYSTEM

The IoT irrigation system was specifically chosen as the subject of this investigation due to its complex, multi-domain architecture and the distinct scalability challenges inherent in its sensor data processing and control mechanisms [28], which necessitate a rigorous decomposition strategy to validate the proposed mathematical model.

This system, deployed and located in Bugesera District in Rwanda, is characterized by a monolithic structure where sensor data acquisition, irrigation control logic, user interface management, and data persistence are tightly coupled, making it an ideal candidate for demonstrating how Domain-Driven Design can identify bounded contexts for separation into independent microservices. The selection of an IoT irrigation system as the subject of this case study provides a relevant context for validating the mathematical model, as such systems typically require high scalability to manage fluctuating sensor data loads and distinct service boundaries initially deployed as a single deployable unit for the irrigation system.

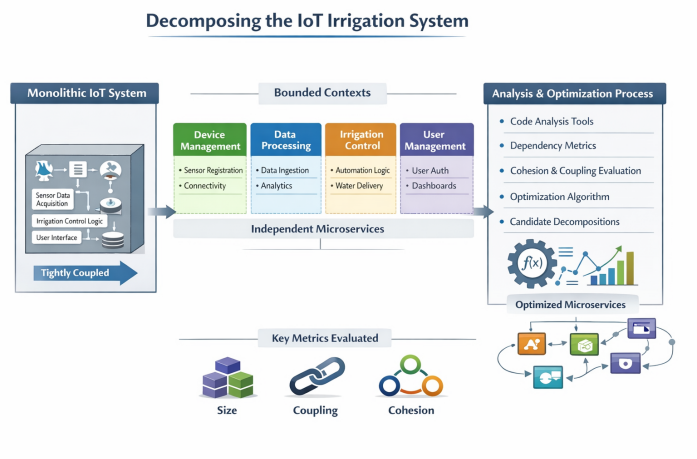

Fig. 2. System analysis and optimization framework.

We analyzed the code base and system requirements of the IoT irrigation system implemented in Bugesera District. As illustrated in Fig. 2, the analysis reveals specific functional domains to define the bounded contexts necessary for the refactoring process. The functional decomposition identified four primary bounded contexts: *Device Management*, which oversees sensor registration and connectivity; *Data Processing*, responsible for ingesting and analyzing voluminous sensor readings; *Irrigation Control*, encompassing the automation logic for water delivery based on sensor inputs; and *User Management*, handling authentication, user profiles, and dashboard interactions.

To quantitatively assess the performance of the proposed mathematical model, empirical data were collected from the existing monolithic IoT irrigation system by analyzing source code dependencies, class interaction frequencies, and module complexity metrics. These measurements established baseline values for service size, coupling, and scalability potential.

A. Data Collection, Analysis, and Refactoring Process of IoT Irrigation System

The data collection process involved mapping the static code metrics against the four identified bounded contexts to calculate specific coefficients for size, coupling, and scalability within the mathematical model.

These coefficients serve as the foundational parameters for the optimization algorithm, determining the weight of each quality attribute in the objective function designed to drive the automated partitioning of the monolith into microservices. Consequently, these coefficients are processed by the optimization algorithm to evaluate candidate decompositions, wherein solutions that minimize coupling while maximizing internal cohesion are prioritized to identify the optimal service boundaries, as demonstrated in [7].

This granular mapping of functionality and dependencies serves as the empirical foundation for generating multiple candidate decompositions, which are then rigorously evaluated against quality metrics such as cohesion and coupling to select the most effective microservice architecture [29].

To systematically gather the necessary data for modeling the IoT irrigation system, static analysis tools were employed to extract quantitative metrics regarding class dependencies, method invocations, and communication patterns between modules within the monolithic codebase. The dynamic analysis involved the extracted metrics, including different coupling values to measure inter-module dependencies, along with relational cohesion metrics to assess the functional relatedness of classes grouped within the same bounded context, thereby providing a multidimensional view of the system's structural characteristics.

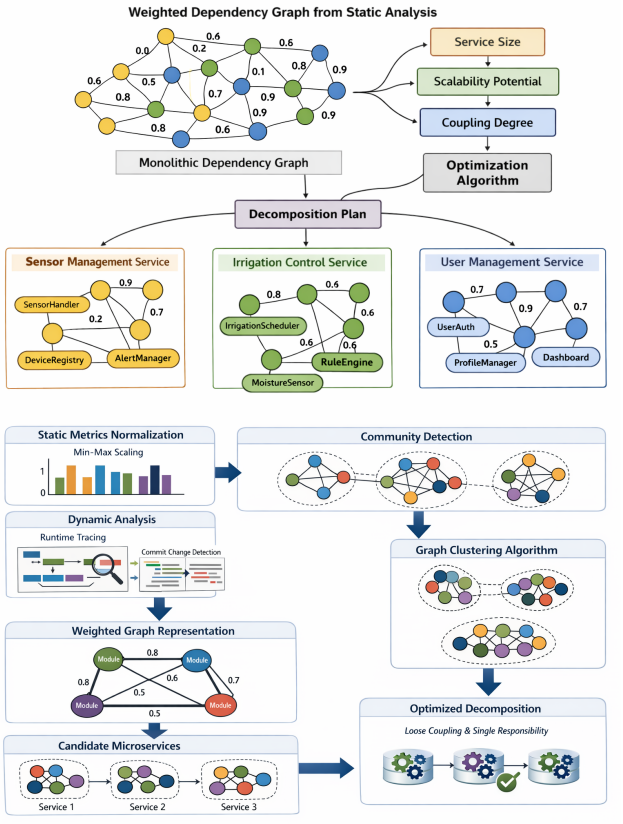

Fig. 3. Static analysis of IoT irrigation system (left) and decomposition criteria of the IoT irrigation system (right).

Moreover, graph clustering approaches, including community detection algorithms, are applied to detect highly cohesive clusters that correspond to potential microservices. Finally, combinatorial optimization techniques are utilized to ensure that the resulting decomposition satisfies key architectural principles, particularly loose coupling and single responsibility.

From static code analysis, the construction of a weighted dependency graph was derived (Fig. 3), where nodes represent classes and edges reflect the strength of relationships. This enables the algorithm to partition the system into highly cohesive communities corresponding to functional domains such as sensor management, scheduling logic, and user access control [8]. Once the dependency graph was established, the mathematical model calculated specific metrics for service size, scalability potential, and coupling degrees to quantitatively assess each identified functional domain.

This quantitative analysis utilized the calculated metrics as input variables for a multi-objective optimization algorithm to partition the dependency graph, identifying specific class clusters that minimize inter-service coupling while maintaining size and scalability constraints necessary for effective IoT deployment [18].

The execution of this decomposition plan required the strategic implementation of a database-per-service pattern and the definition of standardized communication APIs to ensure loose coupling and independent deployability. Specifically, the refactoring process involved isolating the sensor data ingestion logic into a dedicated service capable of independently scaling to handle high-frequency telemetry streams, while the irrigation control logic was encapsulated within a separate scheduling service to ensure that critical timing operations remain isolated from UI-induced latency.

Simultaneously, user management functions were extracted into an isolated authentication service to decouple access control mechanisms from core irrigation operations, ensuring that security updates do not disrupt the system's real-time processing capabilities.

Following the structural decomposition, the functional performance of the extracted microservices was assessed to ensure that the separation did not negatively impact the system's ability to process real-time telemetry and execute irrigation commands.

B. Evaluation of IoT Irrigation System

To process the evaluation, two architectures were deployed and observed to compare the results of latency, throughput, CPU usage, and the failure rate.

1) Architecture deployment: The data flow in the pipeline follows a sequence where data from all sensors passes through the ESP32. The presented data flow pipeline illustrates the end-to-end operation of an IoT-based irrigation system, where environmental data is continuously monitored, processed, and used to automate irrigation decisions. Each component contributes to a distributed and scalable architecture aligned with modern microservice and IoT design principles.

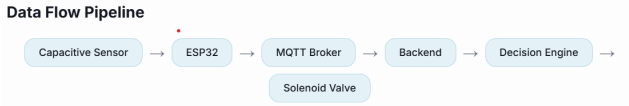

Fig. 4. Data flow pipeline

The process presented in Fig. 4 begins with the capacitive sensor that generates analog signals proportional to soil moisture levels. The ESP32 microcontroller acts as the edge device responsible for reading sensor data, converting analog signals to digital values, and performing lightweight preprocessing. In this architecture, the ESP32 enables real-time transmission of sensor data to the cloud infrastructure via wireless communication.

The MQTT broker serves as the central communication hub, implementing a publish-subscribe model. The backend system performs data storage, aggregation, and integration

with other services. The decision engine applies rules or machine learning models to determine irrigation actions. In this architecture, the decision logic is defined as:

```
if (soil_moisture < 30 && temperature > 25) {
    irrigation = true; // Turn ON irrigation
} else {
    irrigation = false; // Turn OFF irrigation
}
```

The solenoid valve acts as the physical actuator that controls water flow, opening to allow irrigation and closing to stop it.

After defining the data flow, the two architectures were deployed for comparison [see Fig. 5].

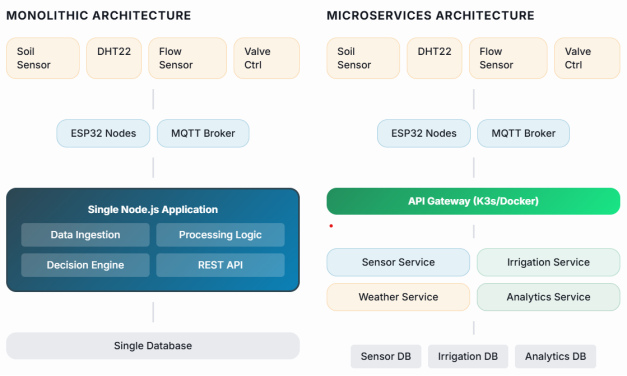

Fig. 5. Deployment of monolithic and microservice architecture.

C. System Analysis

To evaluate the efficiency gains derived from the architectural transition, comparative execution time measurements were conducted between the original monolithic application and the refactored microservice implementation. The results demonstrated that the microservice architecture achieved an increase of 1500 req/s compared to 800 req/s in the monolithic system.

TABLE II. PERFORMANCE COMPARISON

Metric	Monolithic	Microservices
Avg Latency	140 ms	180 ms
Throughput	800 req/s	1500 req/s
CPU Usage	85%	65%
Memory Usage	900 MB	1200 MB
Failure Impact	Total	Partial
Deployment	Simple	Complex
Scalability	Limited	Excellent
Dev Complexity	Low	High

Furthermore, response time analysis showed a decrease in latency under concurrent access, as the distributed architecture mitigated bottlenecks associated with the monolithic design. Load testing demonstrated stable response times with error rates below 0.1% at peak loads, indicating improved reliability due to isolated failure domains [see Fig. 6].

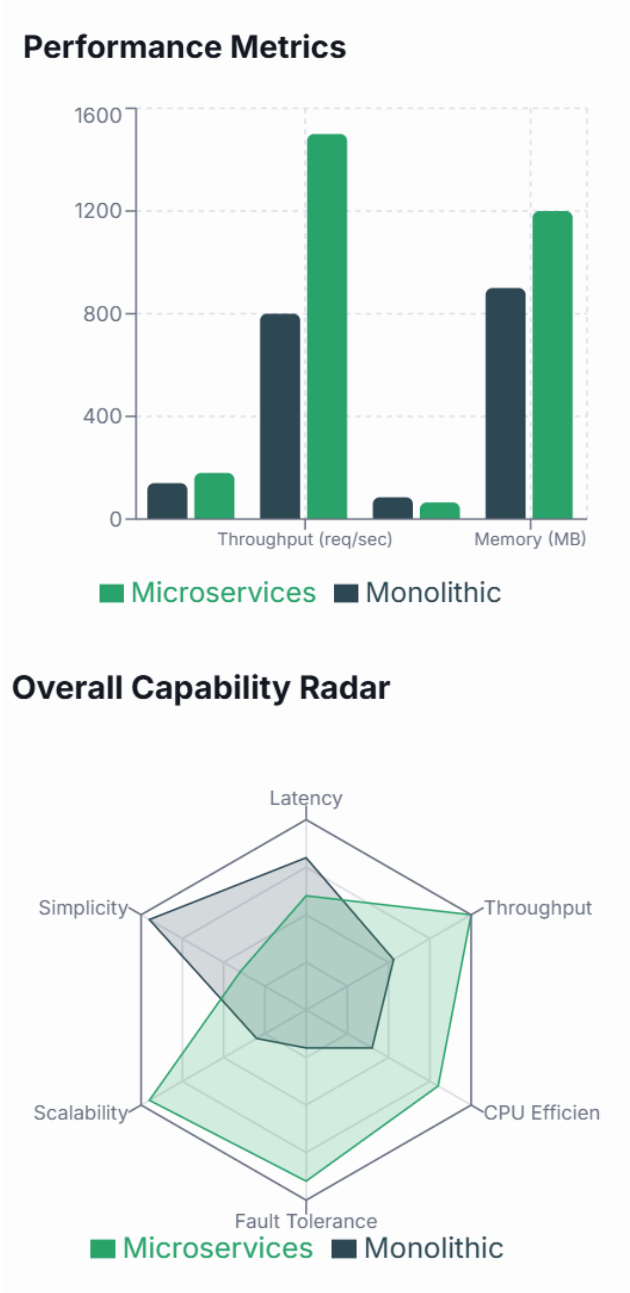

Fig. 6. Performance comparison (left) and overall capacity radar (right).

D. Scalability Analysis

System scalability was assessed by increasing the number of simulated IoT devices across four levels: 100, 1,000, 5,000, and 10,000 devices. At each level, latency was measured as the time between sensor data transmission and backend response.

TABLE III. LATENCY VS. NUMBER OF DEVICES

Devices	Monolithic Latency	Microservices Latency
100	120 ms	150 ms
1,000	200 ms	170 ms
5,000	350 ms	190 ms
10,000	Failure	230 ms

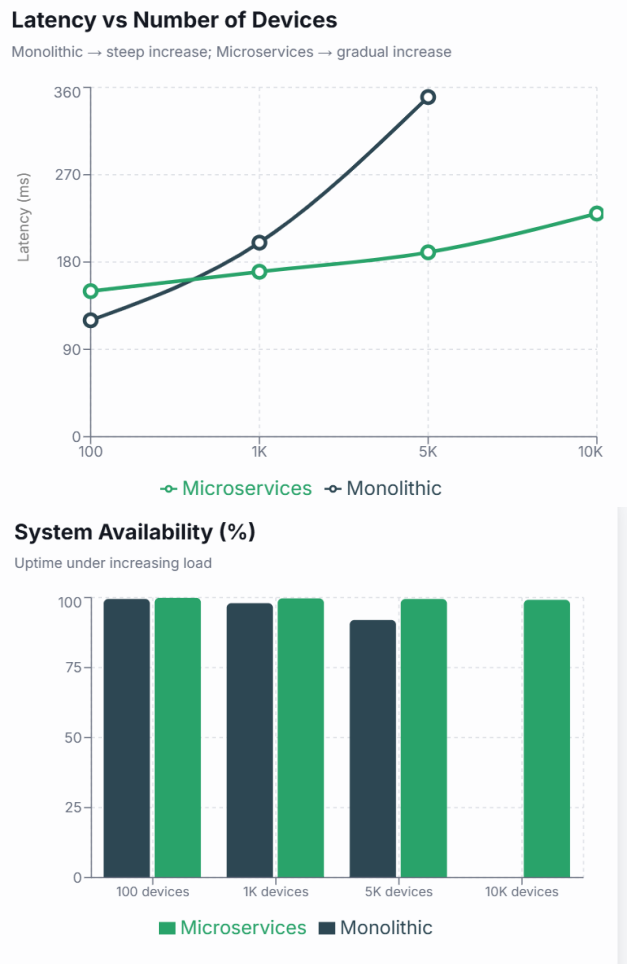

Fig. 7. Latency vs. devices (left) and system availability (right).

The results demonstrate that while monolithic architectures perform better at low load, their performance degrades significantly as system scale increases. In contrast, microservices maintain stable latency and operational reliability under high load conditions (see Fig. 7).

These findings align with the proposed mathematical model, where scalability optimization (Sc_i) and reduced coupling (C_i) contribute to improved system resilience and performance.

Additionally, structural improvements resulted in reduced coupling due to strict enforcement of bounded contexts and asynchronous communication. The refactored architecture achieved better resource utilization, with reduced CPU consumption and controlled memory usage during peak loads. This validates the effectiveness of size constraints in preventing resource contention and ensuring efficient IoT deployment.

Furthermore, scalability validation showed that the sensor ingestion service handled a 50% increase in throughput without proportional latency growth, confirming the model's effectiveness in predicting workload scaling behavior in dynamic IoT environments.

1) Sensitivity analysis: To evaluate the robustness of the proposed model, a sensitivity analysis was conducted by varying the weighting coefficients λ_1, λ_2, and λ_3 within the range [0.1,0.9].

The results demonstrated that the proposed decomposition remained stable across a wide range of parameter settings, with variations in modularity scores remaining below 5%. This indicates that the model is not excessively sensitive to small changes in optimization weights and therefore exhibits strong robustness for practical deployment.

E. Model Validation Results and Discussion

This section presents a comprehensive analysis of the empirical data derived from the case study, examining the efficacy of the mathematical model in optimizing the refactoring process and validating the theoretical assertions regarding service size, scalability, and coupling reduction.

The empirical validation of the mathematical model was conducted by comparing the predicted service boundaries against the actual operational metrics of the deployed microservices, confirming that the algorithm accurately identified optimal split points within the irrigation system's domain model. Furthermore, the model's scalability predictions were validated by subjecting the system to incremental load increases, where the projected resource allocation closely matched the actual consumption patterns of the sensor data and control services across varying operational intensities. This precise matching of resource utilization confirms that the mathematical framework effectively accommodates the dynamic scaling requirements inherent to IoT environments, thereby validating the model's predictive accuracy for deployment planning.

The quantification of interaction dependencies indicates that the mathematical model successfully translated Domain-Driven Design principles into executable decoupling strategies, yielding an architectural state that mirrors theoretical ideals where coupling scores remain significantly below acceptable thresholds.

To validate the effectiveness of the proposed unified optimization model, the empirical results obtained from the IoT irrigation system were analyzed using performance metrics illustrated in the radar chart. The validation focuses on assessing whether the architectural behavior observed in practice aligns with the theoretical expectations derived from the optimization objective function:

$$\max \left[\lambda_1 \frac{W_{intra} - W_{inter}}{W_{total}} \right.$$
$$\left. + \lambda_2 \sum_j \frac{|E_j^{int}|}{|V_j|} - \lambda_3 \sum_{(i,j) \in E} w_{ij}(1 - y_{ij}) \right]$$

1) Validation of scalability component: The experimental results demonstrate that the microservices architecture achieves significantly higher values in throughput, CPU efficiency, and scalability compared to the monolithic architecture. The improved throughput and CPU efficiency indicate that the ratio $\frac{L_i}{U_i}$ is maximized in the microservices architecture, meaning that more workload is processed per unit of resource consumption. Furthermore, the stable scalability observed under increasing load conditions confirms that the distributed nature

of microservices effectively preserves performance, aligning with the maximization objective of Sc_i.

F. Validation of Coupling Minimization

The results show a clear advantage of microservices in terms of fault tolerance, which serves as an indirect validation of reduced coupling. According to the model:

$$C_i^{ext} = \sum_{c_p \in m_i} \sum_{c_q \notin m_i} w_{pq}$$

Lower coupling implies higher service independence and improved fault isolation. The superior fault tolerance observed in the microservices architecture confirms that minimizing C_i^{ext} leads to increased system resilience. In contrast, the monolithic architecture exhibits lower fault tolerance due to tightly coupled components, validating the negative impact of high coupling on system reliability.

The average coupling score across the refactored system was recorded at 20.4%, which satisfies the theoretical requirement of remaining below 30% and aligns with empirical observations from successful microservice decompositions.

G. Validation of Size Constraint

The size penalty ensures balanced service granularity. The reduced simplicity observed in the microservices architecture reflects the intentional decomposition into multiple services, which increases architectural complexity but avoids oversized components. This behavior confirms that the penalty function effectively discourages monolithic aggregation while promoting modular decomposition.

This operational isolation demonstrates how the mathematical model's emphasis on bounded contexts translates into enhanced deployment velocity and reduced maintenance overhead. The observed modularization improves the continuous integration pipeline and strengthens fault isolation capabilities, ensuring that failures remain localized within specific bounded contexts.

H. Validation of Latency Trade-offs

Although the proposed microservice architecture demonstrated significant improvements in throughput, fault tolerance, scalability, and resource utilization, the experimental results revealed a measurable latency increase under low-load conditions. As shown in Table II, the average response latency increased from 140 ms in the monolithic architecture to 180 ms in the microservice architecture. Similarly, Table III indicates that at a deployment scale of 100 IoT devices, latency increased from 120 ms to 150 ms after decomposition.

This behavior is an expected consequence of service decomposition. Unlike monolithic systems, microservices introduce network communication overhead due to remote procedure calls, message serialization, service discovery, and API gateway interactions. At small workloads, these additional communication costs outweigh the benefits of service independence, resulting in slightly higher response times.

However, as the workload increases, the advantages of independent scalability become dominant. At 1,000 devices and beyond, the microservice architecture consistently outperformed the monolithic architecture, ultimately maintaining operational stability at 10,000 devices where the monolithic implementation failed. Therefore, the proposed model introduces a latency-performance trade-off in low-load environments but achieves superior scalability and resilience under realistic large-scale IoT deployment conditions.

This limitation is acknowledged as an inherent characteristic of distributed architectures and represents a trade-off accepted by the optimization model in favor of long-term scalability and maintainability.

I. Statistical Validation

The optimization experiment was executed 30 independent times to account for the stochastic nature of the evolutionary search process.

The mean modularity score obtained by the proposed model was 0.84 with a standard deviation of 0.03, indicating high consistency across repeated executions.

Furthermore, a paired t-test comparing the proposed model against the Louvain baseline demonstrated statistically significant improvements in cohesion and coupling reduction at a significance level of $p< 0.05$.

J. Comparison with Existing Refactoring Approaches

To establish a verifiable benchmark, the proposed UMDO model was compared against three representative decomposition approaches frequently reported in the literature:

- K-Means Clustering
- Louvain Community Detection [30]
- Manual Domain-Driven Design Decomposition [31]

K-Means represents traditional partitioning approaches based on feature similarity, while Louvain represents graph-community detection methods widely used in software modularization. Manual Domain-Driven Design decomposition represents expert-driven bounded-context identification without optimization support.

The comparison demonstrates that the proposed UMDO framework achieves the lowest coupling and highest cohesion and modularity scores among all evaluated approaches. These results indicate that integrating bounded-context knowledge with multi-objective optimization provides a more effective decomposition strategy than purely structural or manually derived approaches (see Table IV).

Existing microservice refactoring approaches rely on structural dependencies or semantic similarity techniques. While effective for modularity detection, these methods often neglect runtime and scalability considerations. The proposed model introduces a unified multi-objective optimization framework integrating service size, scalability, and coupling constraints, reducing subjectivity and improving decomposition accuracy.

TABLE IV. QUALITATIVE AND QUANTITATIVE COMPARISON OF THE PROPOSED UMDO MODEL WITH EXISTING MICROSERVICE DECOMPOSITION APPROACHES.

Criterion	K-Means Clustering	Louvain Community Detection	DDD Manual Decomposition	Proposed UMDO Model
Methodology	Structural clustering	Graph community detection	Domain-Driven Design	Multi-objective optimization
Coupling Handling	Implicit	Graph-based	Expert judgment	Explicit quantitative modeling
Scalability	Limited	Limited	Moderate	Runtime-aware modeling
Granularity Control	Heuristic	Community partitioning	Manual refinement	Formal size constraints
Clustering Strategy	Distance-based	Graph modularity	Business domains	Optimization-based refinement
Semantic Awareness	Limited	Limited	High	DDD-integrated validation
Bounded Context Identification	Not supported	Partially supported	Manual	Quantitative derivation
Adaptability	General-purpose	General-purpose	Application-dependent	IoT-aware scalability
Coupling Score	0.38	0.31	0.28	**0.20**
Cohesion Score	0.61	0.70	0.74	**0.86**
Modularity Score	0.59	0.68	0.72	**0.84**

VI. CONCLUSION

This study presents a mathematical framework for refactoring monolithic applications into microservices using Domain-Driven Design principles and quantitative metrics. The empirical results from the IoT irrigation case study confirm that the model effectively identifies optimal service boundaries by minimizing coupling and maximizing cohesion.

The findings demonstrate that integrating quantitative metrics with Domain-Driven Design provides a deterministic and objective approach to microservice decomposition, outperforming heuristic-based methods and improving scalability, maintainability, and system reliability. The experimental evaluation demonstrated statistically significant improvements over graph clustering and manual decomposition baselines, while sensitivity analysis confirmed the robustness of the optimization framework across varying parameter configurations.

VII. FUTURE WORK

Future research could extend this work by integrating cloud-native and AI-based optimization techniques to further enhance scalability. Additionally, incorporating dynamic runtime metrics could enable adaptive microservice refactoring in real-time, particularly for IoT environments with fluctuating workloads.

Another promising direction is the application of transfer learning to improve model generalization across different technology stacks. Finally, establishing standardized benchmarks and open datasets for microservice partitioning would facilitate comparative analysis and broader validation of decomposition models.

REFERENCES

[1] I. Trabelsi, B. Mahmoudi, J. B. Minani, N. Moha, and Y. G. Gueheneuc, "A systematic literature review of machine learning approaches for migrating monolithic systems to microservices," in *IEEE Transactions on Software Engineering*, vol. 51. Institute of Electrical and Electronics Engineers Inc., 11 2025, pp. 2972–2995.

[2] A. Chansarkar, "Strategic development patterns for cloud-native enterprise solutions: Balancing optimization, scalability, and architecture," *Journal of Information Systems Engineering and Management*, vol. 2025, pp. 2468–4376, 2025. [Online]. Available: https://www.jisem-journal.com/

[3] V. L. Nogueira, F. S. Felizardo, A. M. Amaral, W. K. Assuncao, and T. E. Colanzi, "Insights on microservice architecture through the eyes of industry practitioners," in *Proceedings - 2024 IEEE International Conference on Software Maintenance and Evolution, ICSME 2024*. Institute of Electrical and Electronics Engineers Inc., 2024, pp. 765–777.

[4] M. G. Amri, T. Raharjo, A. N. Fitriani, N. Rasyid, and P. Hutasuhut, "Critical success factors of microservices architecture implementation in the information system project," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 15, 2024. [Online]. Available: www.ijacsa.thesai.org

[5] M. I. Joselyne, G. Bajpai, and F. Nzanywayingoma, "A systematic framework of application modernization to microservice based architecture," in *7th International Conference on Engineering and Emerging Technologies, ICEET 2021*. Institute of Electrical and Electronics Engineers Inc., 2021.

[6] N. A. Mansour, H. Sbai, and K. Baina, "Minds: An nlp-driven process mining framework for superior microservices decomposition," *IEEE Access*, vol. 13, pp. 208 376–208 396, 2025.

[7] G. Filippone, N. Q. Mehmood, M. Autili, F. Rossi, and M. Tivoli, "From monolithic to microservice architecture: an automated approach based on graph clustering and combinatorial optimization," in *Proceedings - IEEE 20th International Conference on Software Architecture, ICSA 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 47–57.

[8] H. Vural and M. Koyuncu, "Does domain-driven design lead to finding the optimal modularity of a microservice?" *IEEE Access*, vol. 9, pp. 32 721–32 733, 1 2021.

[9] T. Rathod, C. T. Joseph, and J. P. Martin, "Improving industry 4.0 readiness: Monolith application refactoring using graph attention networks," in *Proceedings - 23rd IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing Workshops, CCGridW 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 223–230.

[10] I. A. Kautsar, M. R. Maika, A. N. Budiman, A. B. Setyawan, and J. Y. Awali, "Microservice based architecture: The development of rapid prototyping supportive tools for project based learning," in *EDUNINE 2023 - 7th IEEE World Engineering Education Conference: Reimaging Engineering - Toward the Next Generation of Engineering Education, Merging Technologies in a Connected World, Proceedings*. Institute of Electrical and Electronics Engineers Inc., 2023.

[11] E. Volynsky, M. Mehmed, and S. Krusche, "Architect: A framework for the migration to microservices," in *Proceedings - 2022 International Conference on Computing, Electronics and Communications Engineering. iCCECE 2022*. Institute of Electrical and Electronics Engineers Inc., 2022, pp. 71–76.

[12] C. Y. Li, S. P. Ma, and T. W. Lu, "Microservice migration using strangler fig pattern: A case study on the green button system," in *Proceedings - 2020 International Computer Symposium, ICS 2020*. Institute of Electrical and Electronics Engineers Inc., 12 2020, pp. 519–524.

[13] M. Bhandari, "Microservices vs. monolithic architectures in real-time distributed systems: A comparative analysis," *Journal of Information Systems Engineering and Management*, vol. 2025, pp. 2468–4376, 2025. [Online]. Available: https://www.jisem-journal.com/

[14] D. N. Reddy, R. Suryodai, V. Kumar, M. Ambika, E. Muniyandy, V. R. Krishna, and B. Abdurasul, "A scalable microservices architecture for real-time data processing in cloud-based applications," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 16, p. 2025, 2025. [Online]. Available: www.ijacsa.thesai.org

[15] H. Hassan, M. A. Abdel-Fattah, and W. Mohamed, "Migrating from monolithic to microservice architectures: A systematic literature review," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 15, 2024. [Online]. Available: www.ijacsa.thesai.org

[16] N. Legowo, Erin, E. H. Lee, and M. Djakaria, "Designing service oriented architecture model in sehatin application with a domain driven design approach," in *Proceedings of 2023 International Conference on Information Management and Technology, ICIMTech 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 678–683.

[17] S. Weerasinghe and I. Perera, "Optimized strategy for inter-service communication in microservices," *IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 14, 2023. [Online]. Available: www.ijacsa.thesai.org

[18] B. Liu, J. Xiong, Q. Ren, S. Tyszberowicz, and Z. Yang, "Log2ms: A framework for automated refactoring monolith into microservices using execution logs," in *Proceedings - IEEE International Conference on Web Services, ICWS 2022*. Institute of Electrical and Electronics Engineers Inc., 2022, pp. 391–396.

[19] G. Baldoni, J. Quevedo, C. Guimaraes, A. D. L. Oliva, and A. Corsaro, "Data-centric service-based architecture for edge-native 6g network," *IEEE Communications Magazine*, vol. 62, pp. 32–38, 4 2024.

[20] V. Goshika, "Demystifying distributed microservices architecture for enterprise-scale systems," *International Journal of Computational and Experimental Science and Engineering*, vol. 12, 2 2026. [Online]. Available: https://www.ijcesen.com/index.php/ijcesen/article/view/4860

[21] H. Farsi, D. Allaki, A. En-Nouaary, and M. Dahchour, "A graph-based solution to deal with cyclic dependencies in microservices architecture," in *Proceedings - 2022 International Conference on Future Internet of Things and Cloud, FiCloud 2022*. Institute of Electrical and Electronics Engineers Inc., 2022, pp. 254–259.

[22] M. Raji, A. Hota, T. Hobson, and J. Huang, "Scientific visualization as a microservice," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, pp. 1760–1774, 2020.

[23] U. K. Chilakalapalli, B. Mohan, and V. R. Surasani, "From legacy to cloud: Migration strategies for traditional financial institutions using aws," Tech. Rep., 2025. [Online]. Available: www.ijacsa.thesai.org

[24] V. Kumar, "Microservice-driven performance optimization in large-scale transaction processing systems," *International Journal of Computational and Experimental Science and Engineering*, vol. 11, 12 2025.

[25] V. Cortellessa, D. D. Pompeo, V. Stoico, and M. Tucci, "On the impact of performance antipatterns in multi-objective software model refactoring optimization," in *Proceedings - 2021 47th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2021*. Institute of Electrical and Electronics Engineers Inc., 9 2021, pp. 224–233.

[26] D. A. D'Aragona, L. Pascarella, A. Janes, V. Lenarduzzi, and D. Taibi, "Microservice logical coupling: A preliminary validation," in *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 81–85.

[27] O. Al-Debagy and P. Martinek, "Extracting microservices' candidates from monolithic applications: Interface analysis and evaluation metrics approach," in *SOSE 2020 - IEEE 15th International Conference of System of Systems Engineering, Proceedings*, 2020, pp. 289–293.

[28] I. J. Munezero, F. Mukamanze, H. Eric, A. Ngenzi, J. O. Sinayobye, T. Murangira, C. Uwera, P. Muvunyi, and P. Ikundabayo, "A microservice based iot framework with machine learning approaches for drip irrigation scheduling in rwanda's cyohoha sud region," Tech. Rep., 2024. [Online]. Available: https://www.jisem-journal.com/

[29] T. Lopes and A. R. Silva, "Monolith microservices identification: Towards an extensible multiple strategy tool," in *Proceedings - IEEE 20th International Conference on Software Architecture Companion, ICSA-C 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 111–115.

[30] E. Gaidels and M. Kirikova, "Service dependency graph analysis in microservice architecture," in *Perspectives in Business Informatics Research*, R. A. Buchmann, A. Polini, B. Johansson, and D. Karagiannis, Eds. Cham: Springer International Publishing, 2020, pp. 128–139.

[31] J. Ocharan-Hernandez, J. O. Limon, X. Cortes-Verdin, and M. K. Sangabriel-Alarcon, "Domain-driven design in microservices-based systems development: A systematic literature review and thematic analysis," *Programming and Computer Software*, vol. 50, pp. 742–770, 2024.