Microservice Decomposition using Many-Objective Optimization

Vadim Peczyński
Faculty of Electronics, Telecommunication and Informatics
Gdansk University of Technology
Gdansk, Poland
vadim.peczynski@pg.edu.pl

Joanna Szłapczyńska
Faculty of Electronics, Telecommunication and Informatics
Gdansk University of Technology
Gdansk, Poland
joanna.szlapczynska@pg.edu.pl

Abstract— **The literature on Microservices Architecture (MSA) presents numerous approaches for decomposing systems into microservices with the goal of improving quality attributes such as scalability, maintainability, and elasticity. Studies indicate that practitioners typically consider multiple objectives simultaneously when defining service boundaries, making microservice decomposition an inherently multi-objective optimization problem. Within this context, evolutionary algorithms are well suited to exploring trade-offs among competing objectives. This work utilizes the Event Storming graph as the primary input for extracting domain entities and identifying their relationships. By combining semantic analysis of entity names with the structural information embedded in the Event Storming graph, candidate microservice decompositions are generated and optimized using the NSGA-III algorithm. The proposed approach seeks to improve the efficiency and accuracy of deriving microservice boundaries from Event Storming models while providing software architects and decision-makers with a systematic framework for analysing and evaluating decomposition alternatives.**

Keywords- Microservices architecture, MSA, Many-objective optimization, MOO, MSA design

I. INTRODUCTION

The Microservices are built as independently deployable units by fully automated Continuous Integration / Continuous Delivery (CI/CD) pipelines - the centralised management should be kept to bare minimum. They are designed around business capabilities, with each service providing a complete implementation of a specific business function, including the user interface, data persistence layer, and external collaborations. Teams aligned with these business capabilities are cross-functional, bringing together all competencies required for development, such as user experience design, database engineering, and project management [1]. In [2] authors further define Microservice Architecture (MSA) as an architectural style that enables independent deployment, autonomous scaling, and enhanced maintainability of complex software systems.

Architects commonly model both the static structure and the dynamic interactions of a software architecture, including its internal components, to better understand system complexity and external dependencies. Traditionally, modelling approaches such as Business Process Model and Notation (BPMN), Unified Modeling Language (UML), and Domain-Specific Languages (DSLs) have been widely employed for this purpose [3]. However, their adoption in industrial practice has declined, as these approaches often require substantial effort to facilitate effective collaboration and communication with stakeholders beyond the development team. Also in [4], the authors confirm that these methods are becoming increasingly less popular. Intuitive graphical "boxes and lines" approaches are reported to be more widely used for microservices design than UML, with 51% of practitioners preferring them compared to 41.5% for UML. Domain-Specific Languages (DSLs) are even less prevalent, accounting for 27% of usage relative to graphical approaches [4].

According to [5], the primary challenges in migrating to a microservices architecture include identifying appropriate service boundaries and their corresponding bounded contexts, analysing asynchronous communication patterns, and performing event-driven decomposition. As reported in [6], increasing system complexity and scalability challenges are among the primary factors motivating organizations to migrate from microservices back to monolithic architectures, often flaws in the initial service decomposition. Similarly, [7] emphasizes that inappropriate service partitioning can result in substantial technical and operational costs. The selection of suitable decomposition objectives and the partitioning of systems into microservices remain key challenges in the microservices design process [8], [9], [10]. Consequently, evaluating the quality of a microservice decomposition is essential to ensure that services remain sufficiently autonomous and do not become excessively coupled, a condition commonly referred to as "monolithic hell" [11]. To address these challenges, several decomposition strategies have been proposed, including business capability-based decomposition and subdomain-driven decomposition [12]. In parallel, the increasing demand for automated decision support has stimulated the development of numerous approaches, methodologies, and tool-supported frameworks designed to assist software architects in identifying and evaluating appropriate microservice boundaries [13], [14], [15].

To bridge the communication gap between technical teams and business stakeholders and to foster their active involvement

Manuscript received July 07, 2026; revised August 07, 2026; accepted July 13, 2026. Published August 25, 2026.
Issue category: Special Issue on DSD/SEAA 2026 on Works in Progress (WiP) Session, Krakow, Poland, Sept. 2026.
Paper category: Full
DOI: doi.org/10.64552/wipiec.v12i2.146

in software architecture design, collaborative and accessible modelling techniques have emerged. One such technique is Event Storming, introduced by Alberto Brandolini. [16]. Event Storming is a workshop-based technique that uses sticky notes to collaboratively model business workflows, processes, and domain knowledge. According to [17], it can be conducted in three forms:

- Big Picture Event Storming, which explores an entire business domain or value stream;
- Process Modelling, which maps current (as-is) and future (to-be) business processes;
- Software Design Event Storming, which translates business processes into specific software and business scenarios.

Based on a set of simple rules, Event Storming is widely recognized as an effective tool for modelling systems developed using Domain-Driven Design (DDD) [12], [18], [19], [20], [21]. During these workshops, stakeholders, domain experts, and developers participate in facilitated collaborative sessions to explore domain knowledge, and establish a shared understanding of the system under development [22].

The aim of this work is to utilize the Event Storming graph as the primary input for extracting domain entities from the system. Event Storming was selected because it captures domain knowledge in a form that enables the identification of bounded contexts, which serve as the basis for constructing the graph and deriving an initial microservice architecture. By combining semantic analysis of entity names with the relationships identified within the Event Storming graph, candidate microservice decompositions are generated using the NSGA-III optimization algorithm. The proposed approach aims to improve the efficiency and accuracy of deriving microservice boundaries from Event Storming models, providing software architects and decision-makers with a systematic means of analysing and evaluating decomposition alternatives.

The remainder of this paper is organized as follows. The Related Work section provides an overview of the existing literature on multi-objective and many-objective optimization approaches for microservice decomposition. The Proposed Method section introduces the semi-automatic approach developed to address the decomposition problem. Next, the Results section presents the outcomes of the experiment conducted using three Event Storming graphs. The Discussion section analyses and interprets the findings, and finally, the Conclusions section summarizes the main contributions of the study and outlines directions for future work.

II. Related works

The existing literature presents a wide range of approaches for decomposing systems into microservices with the objective of improving key quality attributes such as maintainability, scalability, and elasticity ([23], [24], [25]). Microservice decomposition has been formulated as a multi-objective optimization problem in several recent studies ([26], [27], [28]), where the goal is to identify service boundaries that minimise inter-service coupling while maximizing intra-service cohesion, alongside satisfying constraints related to service granularity and other architectural quality attributes. Furthermore, empirical evidence indicates that practitioners typically consider multiple competing objectives simultaneously when making decomposition decisions, often balancing at least four criteria during the extraction process [8]. Given the complexity and conflicting nature of these objectives, evolutionary algorithms have emerged as an effective solution, enabling the exploration of large search spaces and the identification of high-quality trade-off solutions.

a) Multi-Objective Evolutionary Optimization

Evolutionary Algorithms (EAs) have proven particularly effective for multi-objective optimization problems due to their population-based search strategy, which enables the simultaneous discovery of multiple Pareto-optimal solutions in a single execution [29]. Multi-Objective Evolutionary Algorithm (MOEA) effectiveness is grounded in key concepts such as Pareto dominance, convergence toward the Pareto front, and the preservation of solution diversity across the set of non-dominated alternatives. Early work in this area, such as the Vector Evaluated Genetic Algorithm (VEGA) [30], established the foundations of multi-objective evolutionary optimization; however, it is generally considered less effective for addressing complex problems characterized by large and high-dimensional search spaces. Subsequent algorithms, including NSGA-II and NSGA-III, introduced significant improvements through elitist selection mechanisms and advanced diversity-preservation strategies ([29], [31]). NSGA-II achieves computational efficiency through fast non-dominated sorting and crowded-comparison operators, eliminating the need for fitness-sharing parameters and contributing to its widespread adoption. Building upon these advances, NSGA-III extends the applicability of evolutionary optimization to many-objective problems by incorporating adaptive normalization and a reference-point-based niching mechanism, enabling a more effective distribution of solutions across high-dimensional Pareto fronts [31].

Other prominent multi-objective evolutionary algorithms include the Strength Pareto Evolutionary Algorithm (SPEA) and its successor, SPEA2, both of which utilize an external archive to preserve non-dominated solutions and a strength-based fitness assignment mechanism to evaluate individual quality [32]. SPEA2 further enhances the original approach by introducing an improved density estimation technique and a refined archive truncation procedure, enabling a more accurate assessment of solution diversity and a better preservation of boundary solutions along the Pareto front. These improvements contribute to a more balanced distribution of solutions and improved convergence toward high-quality trade-off alternatives.

Multi-objective Evolutionary Algorithm based on Decomposition (MOEA/D) offers an alternative strategy for multi-objective optimization by decomposing a problem into a

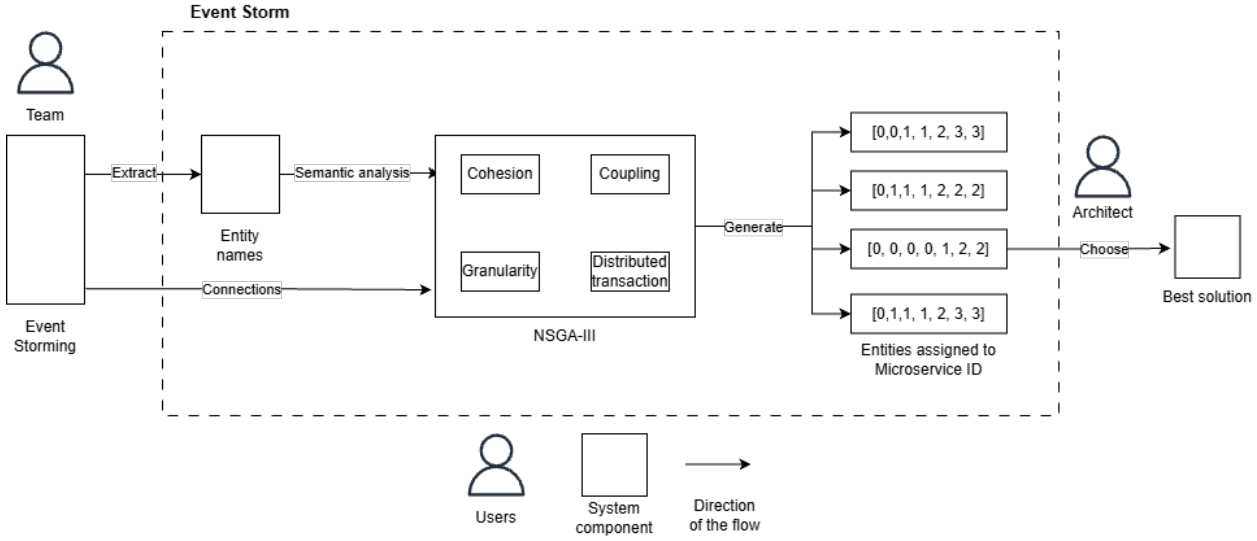

Fig. 1 Overview of the approach for MS decomposition

set of scalar optimization subproblems, each associated with a predefined weight vector [33]. This decomposition enables neighbouring subproblems to share information during the search process, improving computational efficiency and scalability. Despite these advantages, MOEA/D may encounter difficulties when the Pareto front is highly irregular or unevenly distributed, which can limit its ability to maintain a well-spread set of solutions [31].

In contrast to dominance-based approaches, the Indicator-Based Evolutionary Algorithm (IBEA) employs quality indicators to guide the selection process, enabling the direct assessment of solution quality within the objective space [34]. Common variants, such as IBEAε+ and hypervolume-based IBEA (IHD), provide a flexible framework for incorporating decision-maker preferences while eliminating the need for explicit diversity-preservation mechanisms. Fitness values are assigned by evaluating each solution relative to all other individuals in the population, allowing selection pressure to be driven by the chosen performance indicator. Experimental results reported in [34] demonstrate that IBEA consistently outperforms established algorithms such as NSGA-II and SPEA2 across a wide range of benchmark problems, exhibiting strong convergence properties, greater generality, and improved scalability with respect to population size.

b) Algorithms used for Microservice Decomposition

Among the evolutionary algorithms applied to the microservice decomposition problem, NSGA-II is one of the most widely adopted due to its ability to generate diverse sets of Pareto-optimal decomposition alternatives. Its effectiveness in balancing competing architectural objectives has led to its application in a variety of contexts, including cloud migration scenarios, where optimization must account for constraints such as network communication overhead and hosting costs [15], [23], [35]. To address scalability issues associated with large and complex systems, [36] proposes MSExplorer, which reduces the search space through an initial clustering phase prior to performing NSGA-II-based optimization. Beyond microservice decomposition, recent studies have addressed deployment and post-design challenges. Specifically, [37] proposes Yonga, an adaptive placement strategy for resource-constrained environments, while [38] introduces a feature model extraction approach to improve reuse, variability management, and configurability in microservice-based software product lines.

NSGA-III, the successor to NSGA-II, has been applied to complex software decomposition problems involving a larger number of optimization objectives, with implementations reported in [14], [23]. Its reference-point-based selection mechanism makes it particularly suitable for many-objective optimization. Similarly, [39] modelling microservice identification as a many-objective problem and showing that simultaneously optimizing coupling, cohesion, feature modularization, reuse, and network overhead produce higher-quality architectural solutions.

The Indicator-Based Evolutionary Algorithm (IBEA) has also been successfully applied to the microservice decomposition problem through the MSExtractor framework [23]. The reported results indicate that IBEA consistently outperforms NSGA-II and SPEA2 by generating decomposition solutions that provide a more favourable balance between coupling, cohesion, and service granularity. Beyond traditional evolutionary algorithms, hybrid approaches combining Genetic Algorithms with Fuzzy Cognitive Maps have been proposed to capture uncertainty and complex dependencies during decomposition [27], [40]. More recently, researchers have explored combinatorial optimization for automated microservice synthesis [41] and deep reinforcement

learning techniques to derive optimal microservice architectures [35]. In the domain of Software-as-a-Service (SaaS) systems, [33] highlights the importance of variability management through the SAMIM approach, which applies MOEA/D to optimize trade-offs among component commonality, service granularity, and alignment with business objectives.

The decision to create Microservices Architecture (MSA) system is inherently complex and often requires support from dedicated decision-making frameworks and tools. Approaches such as those proposed in [24] facilitate evidence-based decision making by leveraging objective system data, while Multi-Layer Fuzzy Cognitive Maps (MLFCMs) enable scenario analysis and improve the transparency and interpretability of the decision process [27]. The quality of decomposition results is also highly influenced by the configuration of optimization algorithms, making hyperparameter tuning an important factor in achieving effective solutions [42]. Furthermore, weighted loss functions provide a mechanism for incorporating business priorities directly into the optimization process, allowing decomposition outcomes to better align with organizational goals [43]. Given the dynamic nature of software systems, implementation should be viewed as an iterative process that requires continuous monitoring and architectural refinement in response to evolving business requirements and technical constraints [44], [45].

III. Proposed Framework – Event Storm

This section introduces the proposed framework, Event Storm, and subsequently describes the problem formulation and the modifications made to adapt the multi-objective optimization algorithm to the microservice decomposition task. Event Storm utilizes the Event Storming graph to extract domain entities and identify bounded contexts that define an initial microservice architecture. Candidate microservice decompositions are generated by combining semantic analysis of entity names with graph relationships using the NSGA-III optimization algorithm, providing systematic support for evaluating decomposition alternatives.

A. Overview of the method

Event Storming facilitates collaboration between technical and non-technical stakeholders, enabling the collaborative elicitation of domain knowledge. The resulting Event Storming model provides architects with information about domain entities, aggregates, and bounded contexts. Building on this foundation, we extend the Event Storming process by enabling the automated extraction of microservices from an Event Storming graph. This task is formulated as a combinatorial optimization problem, in which entities identified within the graph are partitioned into candidate microservices. Given the large number of possible decompositions, the problem can be modelled as a graph partitioning task, which is known to be NP – hard problem [14], [28]. Consequently, a meta-heuristic search strategy is employed to identify near-optimal solutions within a reasonable computational time. As illustrated in Fig. 1, the proposed Event Storm approach utilizes NSGA-III to explore the search space and generate alternative sets of microservice candidates.

B. Problem formulation

In this work, a microservice architecture $M$is defined as a decomposition of a set of entities E into a collection of microservices, where each microservice represents a cohesive container of related entities. Let $E = \{e_1, e_2, \dots, e_n\}$ denote the set of identified entities in a software system, where n is the total number of entities. The set of candidate microservices is represented as $M = \{m_1, m_2, \dots, m_k\}$, where k denotes the number of microservices and each microservice is assigned a unique identifier. The size of a microservice, denoted as $size(m)$, is defined as the number of entities assigned to it.

A decomposition solution is encoded as a vector of decision variables $X = \{x_1, x_2, \dots, x_n\}$, where $x_i = j$indicates that entity $e_i$is assigned to microservice m_j. For example, the solution $X = \{1,1,2,2,3,3,4,4\}$ represents a decomposition of eight entities into four microservices. Since entities interact to support the required system functionality, dependencies naturally exist among them. Therefore, an effective decomposition should maximize cohesion within microservices while minimizing inter-service coupling and the number of distributed transactions. To support these objectives, the proposed Event Storm approach employs structural and semantic similarity measures to quantify dependencies among entities, guiding the optimization process toward highly cohesive and loosely coupled microservice boundaries.

a) Structural similarity

Each entity $e_i \in E = \{e_1, e_2, \dots, e_n\}$is represented by a structural feature vector $\mathbf{s}_i = [s_{i1}, s_{i2}, \dots, s_{in}]$, where each element s_{ij}denotes the number of structural interactions (connections) between entity $e_i$and entity e_j. In this representation, the vector encodes the connectivity profile of an entity within the system, capturing how it interacts with all other entities in the system. For instance, a vector such as $[0, 0, 0, 4, 0, 0, 0, 1, 4, 1, 0, 1, 2]$indicates that the corresponding entity has four interactions with e_4, one interaction with e_8, four interactions with e_9, and additional interactions with other entities as specified by non-zero entries. Based on these structural representations, the structural similarity between two entities can be calculated:

$$sim_{str}(e_i, e_j) = \sum_{r \in R} \frac{N_r(e_i, e_j)\text{-}min(N_r)}{max(N_r)\text{-}min(N_r)} \in [0,1]$$

where:

- $N_r(e_i, e_j)$ is the number of times entities $e_i$and $e_j$appear as connected elements in process r
- R is the set of all processes,

This approach enables the identification of entities that exhibit similar structural roles, even if they are not directly connected, thereby supporting the detection of cohesive groups during the microservice decomposition process.

b) Semantic similarity

Semantic similarity captures conceptual and domain-level relationships between entities and is computed using vector representations derived from spaCy embeddings. Each entity e_i is represented by a semantic vector $\mathbf{s}_i$, and the similarity between two entities is defined using cosine similarity:

$$\text{sim}_{sem}(e_i, e_j) = \frac{\mathbf{s}_i \cdot \mathbf{s}_j}{\| \mathbf{s}_i \| \ \| \mathbf{s}_j \|} \in [0,1],$$

where ·denotes the dot product and ‖·‖is the Euclidean norm. This measure reflects the conceptual proximity of entities in the embedding space, enabling the capture of domain-level similarity beyond exact lexical matches.

c) Overall similarity

The overall similarity between two entities combines both structural and semantic aspects. It is defined as the sum of string-based similarity and semantic similarity:

$$\text{sim}(e_i, e_j) = 0.5 * \text{sim}_{str}(e_i, e_j) + 0.5 * \text{sim}_{sem}(e_i, e_j) \in [0,1]$$

This aggregated measure integrates lexical overlap with embedding-based semantic relationships, providing a more comprehensive similarity assessment for entity comparison.

C. MOEA adoption

The Event Storm evaluates a candidate microservice decomposition using a multi-objective fitness function that assesses structural quality, communication cost, and architectural balance. Each solution is encoded as a vector $X = \{x_1, x_2, \dots, x_n\}$, where each element x_i assigns entity e_i to a specific microservice. The evaluation procedure operates on a normalized representation of this assignment and computes four complementary objectives: cohesion, coupling, granularity, and distributed transaction cost.

1) Solution representation

Given an input solution X, the algorithm first applies a normalization step to ensure consistent labeling of microservices. The normalized vector X' induces a partition of entities into a set of clusters $M = \{m_1, m_2, \dots, m_k\}$, where k denotes the number of distinct microservices in the solution.

2) Initial population

The initial population is constructed using a random initialization strategy. Given a predefined maximum number of microservices (n), each entity in the application is randomly assigned to a microservice, resulting in a set of candidate decompositions. To ensure the generation of meaningful service boundaries, a constraint parameter (minSize) is introduced, representing the minimum number of classes permitted within a microservice. This constraint prevents the creation of undersized services and contributes to the generation of more balanced and practically viable decomposition solutions.

3) Objective functions

The quality of a candidate microservice decomposition is evaluated through a multi-objective fitness function that incorporates both optimization objectives and feasibility constraints. This fitness function provides a quantitative measure for assessing and comparing alternative decomposition solutions. Each objective corresponds to a distinct architectural quality attribute and is formulated to be either minimized or maximized according to the desired system characteristics. Consequently, the search process aims to identify solutions that achieve an optimal balance among potentially conflicting objectives while adhering to all specified constraints. In this work, the optimization framework focuses on the following four objectives:

a) Cohesion (maximize)

Cohesion measures the degree of functional and structural relatedness of entities within the same microservice. It is computed using a cohesion function defined for each microservice, following the formulation in [28] and adapted to measure entity similarity:

$$Coh(m) = 1 - \frac{\sum_{e_i, e_j \in m, e_i \neq e_j} sim(e_i, e_j)}{\frac{| m | (| m | - 1)}{2}}$$

where:

- $sim(e_i, e_j)$ is the similarity between entities e_i and e_j,
- $| m |$ is the number of entities in microservice m.

Then for overall cohesion:

$$Cohesion(\mathcal{M}) = 1 - \frac{\sum_{m_i \in \mathcal{M}} Coh(m_i)}{| \mathcal{M} |}$$

where:

- $Coh(m_i)$ is the cohesion value of microservice m_i,
- $| \mathcal{M} |$ is the total number of microservices.

Higher cohesion values indicate stronger intra-service relationships. In the optimization process, cohesion is maximized.

b) Coupling (minimize)

Coupling quantifies the level of interdependencies between different microservices. It is computed following the formulation in [28], with adaptations to account for entity similarity, as follows:

$$Cou(m_1, m_2) = \frac{\sum_{e_i \in m_1,\, e_j \in m_2} sim(e_i, e_j)}{| m_1 | \cdot | m_2 |}$$

where:

- $sim(e_i, e_j)$is the similarity between entities $e_i$and e_j,
- $| m_1 |$and $| m_2 |$denote the sizes of the two microservices.

Lower coupling values are preferred, as they indicate reduced inter-service communication and dependency.

c) Granularity (maximize)

Granularity captures the decomposition balance in terms of the number of generated microservices relative to the number of entities. The objective function is a normalized and modified version of the objective function proposed in [28], formulated as follows::

$$G(X') = \frac{k}{| E |}$$

where:

- $k$is the number of detected microservices and
- $| E |$is the total number of entities.

This objective promotes controlled decomposition and avoids extreme fragmentation.

d) Distributed Transactions Cost (minimize)

Distributed transaction cost evaluates the communication overhead induced by inter-service interactions. It is computed based on the execution paths or dependency graph PE:

$$DT_{cost} = \frac{\sum_{r \in R} Jump\left(a_{u_i} \neq a_{u_{i+1}}\right)}{\sum_{r \in R}(| U_r | - 1)}$$

where:

- a_{u_i} is microservice associated with entity u_i
- $\mathbf{Jump}(A) = \begin{cases} 1, & \textit{if A is true} \\ 0, & \textit{otherwise} \end{cases}$
- $| U_r |$= number of entities in process r
- R is the set of all processes

This objective penalizes decompositions that require frequent cross-service transactions.

4) Constraints

To ensure meaningful microservice formation, a minimum size constraint is enforced for each cluster:

$$\min_{m_i \in M} | m_i | \geq \text{min\_size}$$

If any microservice violates this constraint, the solution is penalized accordingly:

$$G_{constraint}(X') = \text{min\_size} - \min\left(| m_1 |, | m_2 |, \ldots, | m_k |\right)$$

5) Multi-objective Optimization Formulation

The final fitness vector is defined as:

$$F(X') = [-C_{oh}(X'), C_{ou}(X'), DT(X'), -G(X')]$$

The optimization problem simultaneously minimizes all objectives, encouraging decompositions that exhibit high cohesion, low coupling, low transaction cost, and balanced granularity. In the PyMoo framework, all objective functions are formulated as minimization problems. Consequently, cohesion and coupling metrics are multiplied by −1 to ensure consistency with this minimization requirement.

IV. Experiments & Results

This section presents the experimental evaluation of the proposed approach for automated microservice decomposition based on Event Storming graphs. The study investigates how different architectural preference profiles influence the resulting decompositions and assesses the stability and quality of the generated solutions under repeated optimization runs.

a) Experiment

The proposed approach was evaluated using three Event Storming graphs representing different software systems. The experiment involved two hypothetical software architects with contrasting design preferences, whose objective configurations were used to guide the optimization process. The first architect represents a monolithic-oriented perspective, prioritizing cohesion and distributed transaction cost, thereby favoring more strongly integrated services. The second architect represents a microservice-oriented perspective, emphasizing coupling and granularity to promote loosely coupled and finer-grained services. For each Event Storming graph, the optimization process was executed 15 times to account for the stochastic nature of the NSGA-III algorithm.

b) Results

The resulting Pareto-optimal solutions were ranked using the TOPSIS method to select the most preferred decomposition in each run. This enables a consistent selection of representative solutions from the Pareto front for subsequent analysis. For one of the evaluated Event Storming graphs, the mean objective values of the selected solutions illustrate the effect of the differing preference profiles. The monolithic-oriented configuration achieved higher cohesion reflecting a stronger focus on intra-service relationships. In contrast, the microservice-oriented configuration resulted in lower coupling, higher granularity, and reduced distributed transaction cost, indicating a shift towards more independent and fine-grained services.

The quality of the obtained Pareto fronts is assessed using the hypervolume (HV) indicator, which measures the volume

of the objective space dominated by the Pareto set relative to a predefined reference point. In this study, all objectives are normalized to the range $[0, 1]$, and the reference point is defined as $r = (1.1, 1.1, 1.1, 1.1)$. The hypervolume is therefore computed with respect to this reference point, where higher values indicate better convergence towards the Pareto-optimal region and improved diversity of solutions.

The maximum achievable hypervolume corresponds to the ideal case in which the Pareto front reaches the origin $(0, 0, 0, 0)$, yielding $HV_{\max} = (1.1 - 0)^4 = 1.1^4 = 1.4641$. Accordingly, the valid range of the hypervolume indicator is $0 \leq HV \leq 1.4641$. In the conducted experiments, a hypervolume value of approximately 0.95 was obtained, which corresponds to about 68% of the theoretical maximum. This indicates that the proposed method is robust with respect to varying architectural priorities, while still enabling controlled steering of the decomposition outcome according to stakeholder preferences. The results are presented in Table 1.

TABLE I. RESULTS OF THE EXPERIMENTS

Scenario	**Results**		
	Metric	***User 1 (Monolithic)***	***User 2 (Microservice)***
1	Cohesion (mean)	0.2366	0.0336
	Coupling (mean)	0.0799	0.0901
	Granularity (mean)	0.3333	0.9000
	DT Cost (mean)	0.1334	0.7619
	Hypervolume (mean)	0.9646	
2	Cohesion (mean)	0.3045	0.04774
	Coupling (mean)	0.1582	0.16584
	Granularity (mean)	0.2778	0.87778
	DT Cost (mean)	0.3281	0.92982
	Hypervolume (mean)	0.9541	
3	Cohesion (mean)	0.2532	0.0615
	Coupling (mean)	0	0.2319
	Granularity (mean)	0.1667	0.8476
	DT Cost (mean)	0	0.9111
	Hypervolume (mean)	1.0534	

V. DISCUSSION

The following section describe the key design decisions underpinning the proposed approach and provide a discussion of the experimental results. Presented framework for microservice extraction operates on the Event Storming graph. Cards are grouped at the entity level, although a finer-grained (card-level) decomposition is also possible. The quality of the resulting microservice decomposition is strongly dependent on the quality of the input Event Storming graph. Therefore, a

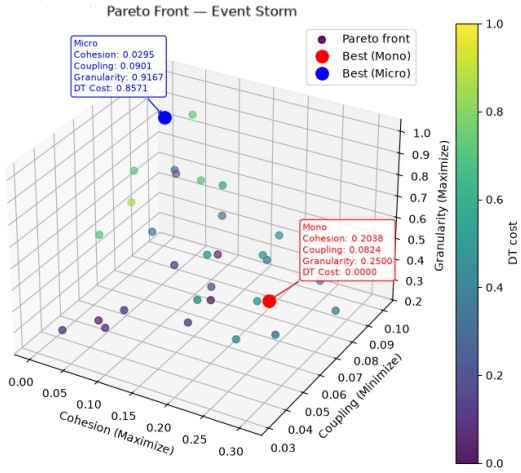

Fig. 2 Representative results for User 1 (red) and User 2 (blue) for Scenario 1

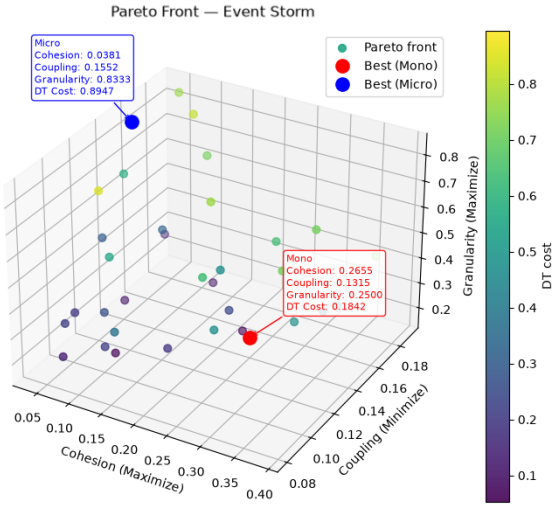

Fig. 3 Representative results for User 1 (red) and User 2 (blue) for Scenario 2

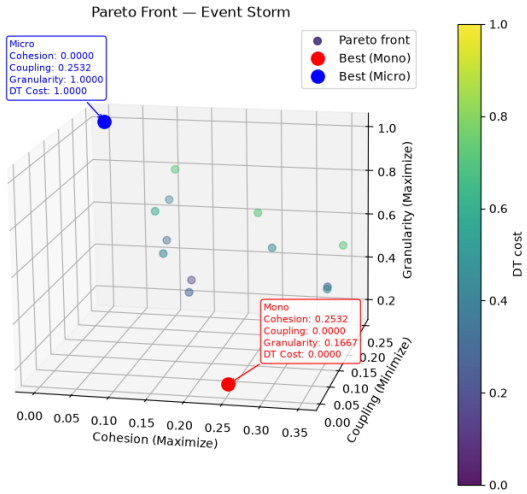

Fig. 4 Representative results for User 1 (red) and User 2 (blue) for Scenario 3

preliminary refinement phase is required, including the simplification of business processes, as well as the removal of redundant or unused cards, to ensure a consistent and high-quality representation of the domain. This step is essential, as microservices are expected to be small, cohesive, and functionally meaningful units.

A multi-objective formulation is necessary, as relying solely on coupling may lead to overly fine-grained decompositions (e.g., single-entity services), whereas optimizing only cohesion may collapse the solution into a single monolithic service. Initial experiments further indicated that these objectives alone tend to produce extreme solutions. To address this limitation, additional objectives related to granularity and distributed transaction cost are introduced to better control service size and enforce user-defined constraints.

The goal of the approach is to group entities that are both structurally and semantically related into candidate microservices. The evolutionary optimization process progressively favors solutions that satisfy these objectives, while weaker candidates characterized by low cohesion or high distributed transaction cost are eliminated over successive generations. Nevertheless, the generation of a deployable microservice architecture requires human involvement for Event Storming preparation, validation, and post-processing refactoring, which is outside the scope of this work.

The inter-entity dependencies are inferred using structural and semantic similarity measures, which act as heuristics for capturing latent relationships in the absence of explicit use-case information. While this enables automation in incomplete or informal modeling settings, it also introduces the risk of grouping entities that are syntactically or semantically similar but not necessarily functionally related.

The results in Table 1 demonstrate that the proposed optimization framework successfully adapts the generated decompositions to different stakeholder preferences while maintaining comparable optimization quality.

Across all scenarios, the monolithic-oriented configuration consistently produced substantially higher cohesion than the microservice-oriented configuration. In contrast, the microservice-oriented configuration achieved significantly higher granularity, indicating the generation of smaller and more fine-grained microservices. This behavior is consistent with the optimization objectives assigned to each preference profile.

The remaining objectives further illustrate the trade-offs between the two architectural styles. In Scenarios 1 and 2, coupling values are comparable for both configurations, with the monolithic-oriented profile producing slightly lower values. In Scenario 3, however, the monolithic-oriented configuration achieved zero coupling and distributed transaction cost, indicating a decomposition with no inter-service communication. The microservice-oriented configuration exhibited higher coupling (0.2319) and distributed transaction cost (0.9111), reflecting the increased communication overhead typically associated with finer-grained service decomposition.

The hypervolume values remained consistently high across all scenarios, ranging from 0.9541 to 1.0534. This indicates that the proposed NSGA-III-based optimization approach produced well-converged and diverse Pareto fronts regardless of the selected architectural preference profile. Overall, the results confirm that modifying the optimization objectives effectively steers the decomposition process toward either monolithic-oriented or microservice-oriented solutions while maintaining high optimization quality. Figures 1–3 present representative optimization results for the three experimental scenarios. The figures illustrate a consistent distinction between the proposed solutions generated for User 1 and User 2, reflecting the different architectural preference profiles across all experiments.

VI. Threats to validity

Threats to internal validity relate to factors within the study that may influence the results. Due to the stochastic nature of the proposed evolutionary approach, some variability in outcomes is expected. To mitigate this, each experiment was repeated 15 times for each user, and mean values were used for analysis. The comparison relied on the Hypervolume (HV) indicator, which, while widely used due to its Pareto compliance, may not fully capture all aspects of performance when used in isolation. Moreover, without knowledge of the true Pareto Front, convergence metrics such as Generational Distance (GD) and Inverted Generational Distance (IGD) cannot be reliably computed. As any approximation would introduce bias, HV was selected as the primary indicator; however, future work should incorporate additional metrics to better capture convergence and diversity.

Threats to construct validity arise from the chosen evaluation metrics. Although cohesion, coupling, granularity, and distributed transaction cost provide multiple perspectives on decomposition quality, they may not capture all relevant aspects. Future studies could integrate additional quality measures to improve evaluation comprehensiveness.

Threats to external validity are mainly related to limited generalizability. The evaluation was conducted on three Event Storming graphs, which constrains broader applicability. Additionally, qualitative assessment was performed only by the authors, without input from external experts. Future work will involve practitioner studies and comparative evaluations with existing microservice decomposition approaches to strengthen external validity.

VII. Conclusions

In this paper, we presented Event Storm, a novel framework for microservice extraction that formulates the decomposition problem as a multi-objective combinatorial optimization task. The proposed method leverages the NSGA-III algorithm to search for optimal decompositions of Event Storming graphs

while simultaneously considering both structural and semantic relationships among identified entities. The evaluation results demonstrate that Event Storm is capable of generating microservice decompositions characterized by high cohesion and low coupling, two key quality attributes of effective microservice architectures.

As part of our future work, we plan to investigate alternative optimization algorithms and compare the proposed approach with existing state-of-the-art microservice decomposition techniques. We also intend to conduct a more extensive evaluation involving software architects and developers, as well as a broader set of case studies, to further assess the practical applicability of the approach. In addition, future research will incorporate non-functional quality attributes that play a critical role in microservice architectures, including scalability, cost efficiency, and testability, thereby enabling a more comprehensive assessment of decomposition quality.

Furthermore, we plan to explore preference-based multi-objective optimization techniques, particularly the integration of the w-dominance relation[46] into the decomposition process. Unlike traditional Pareto dominance, w-dominance enables the incorporation of decision-maker preferences and associated uncertainties through weight intervals that express the relative importance of optimization objectives. This approach would allow software architects to explicitly prioritize architectural concerns according to project-specific requirements without requiring prior knowledge of expected solutions. By guiding the search process toward solutions that better reflect stakeholder preferences, w-dominance has the potential to improve the relevance and practical usefulness of the generated microservice decomposition alternatives.

DATA AVAILABILITY

Data will be made available on request.

REFERENCES

[1] J. Lewis and M. Fowler, "Microservices - a definition of this new architectural term," 2014.

[2] L. Qian, J. Li, X. He, R. Gu, J. Shao, and Y. Lu, "Microservice extraction using graph deep clustering based on dual view fusion," *Inf. Softw. Technol.*, vol. 158, p. 107171, 2023, doi: https://doi.org/10.1016/j.infsof.2023.107171.

[3] O. Zimmermann, K. Luban, M. Stocker, and G. Bernard, "Continuous process model refinement from business vision to event simulation and software automation," in *Proceedings of the 5th International Workshop on Software-intensive Business: Towards Sustainable Software Business*, ACM, May 2022, pp. 59–66. doi: 10.1145/3524614.3528631.

[4] M. Waseem, P. Liang, M. Shahin, A. Di Salle, and G. Márquez, "Design, monitoring, and testing of microservices systems: The practitioners' perspective," *Journal of Systems and Software*, vol. 182, p. 111061, 2021, doi: https://doi.org/10.1016/j.jss.2021.111061.

[5] H. Ünlü, D. E. Kennouche, G. K. Soylu, and O. Demirörs, "Microservice-based projects in agile world: A structured interview," *Inf. Softw. Technol.*, vol. 165, p. 107334, 2024, doi: https://doi.org/10.1016/j.infsof.2023.107334.

[6] R. Su, X. Li, and D. Taibi, "Back to the Future: From Microservice to Monolith," May 2023, doi: 10.48550/arXiv.2308.15281.

[7] Y. Wei, Y. Yu, M. Pan, and T. Zhang, "A Feature Table approach to decomposing monolithic applications into microservices," *ACM International Conference Proceeding Series*, pp. 21–30, May 2020, doi: 10.1145/3457913.3457939.

[8] L. Carvalho, A. Garcia, W. K. G. Assunção, R. de Mello, and M. de Lima, "Analysis of the Criteria Adopted in Industry to Extract Microservices," in *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*, May 2019, pp. 22–29. doi: 10.1109/CESSER-IP.2019.00012.

[9] T. Lopes and A. R. Silva, "Monolith Microservices Identification: Towards An Extensible Multiple Strategy Tool," in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2023, pp. 111–115. doi: 10.1109/ICSA-C57050.2023.00034.

[10] V. Peczyński, J. Szłapczyńska, and A. Szopińska, "Patterns in Design of Microservices Architecture: IT Practitioners' Perspective," *WiPiEC Journal - Works in Progress in Embedded Computing Journal*, vol. 11, no. 1, p. 10, Sep. 2025, doi: 10.64552/wipiec.v11i1.101.

[11] S. Santos and A. R. Silva, "Microservices Identification in Monolith Systems: Functionality Redesign Complexity and Evaluation of Similarity Measures," *Journal of Web Engineering*, vol. 21, no. 5, pp. 1543–1582, May 2022, doi: 10.13052/jwe1540-9589.2158.

[12] C. Richardson, *Microservices Patterns: With examples in Java*. Manning, 2018. [Online]. Available: https://books.google.pl/books?id=UeK1swEACAAJ

[13] D. Bajaj, U. Bharti, I. Gupta, P. Gupta, and A. Yadav, "GTMicro—microservice identification approach based on deep NLP transformer model for greenfield developments," *International Journal of Information Technology (Singapore)*, vol. 16, no. 5, pp. 2751–2761, Jun. 2024, doi: 10.1007/S41870-024-01766-5.

[14] T. Kinoshita and H. Kanuka, "Enhancing Automated Microservice Decomposition via Multi-Objective Optimization," *IEEE Access*, vol. 12, pp. 55697–55710, Apr. 2024, doi: 10.1109/ACCESS.2024.3389700.

[15] K. Sellami, M. A. Saied, and A. Ouni, "A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications," in *Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering*, in EASE '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 201–210. doi: 10.1145/3530019.3530040.

[16] A. Brandolini, "Introducing Event Storming," 2013.

[17] E. van Kelle, G. Verschatse, and K. Baas-Schwegler, *Collaborative Software Design: How to facilitate domain modeling decisions*. Simon and Schuster, 2025.

[18] S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, Incorporated, 2019. [Online]. Available: https://books.google.pl/books?id=iul3wQEACAAJ

[19] V. Vernon, *Domain-driven Design Distilled*. Addison-Wesley, 2016. [Online]. Available: https://books.google.pl/books?id=h0u7jwEACAAJ

[20] V. Khononov, *Learning Domain-Driven Design*. O'Reilly Media, Inc., 2021.

[21] S. R. Goniwada, *Cloud Native Architecture and Design*. Apress, 2022. doi: 10.1007/978-1-4842-7226-8.

[22] H. Gardner, A. F. Blackwell, and L. Church, "The patterns of user experience for sticky-note diagrams in software requirements workshops," *J. Comput. Lang.*, vol. 61, p. 100997, 2020, doi: https://doi.org/10.1016/j.cola.2020.100997.

[23] I. Saidani, A. Ouni, M. W. Mkaouer, and A. Saied, "Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11895 LNCS, SPRINGER INTERNATIONAL PUBLISHING AG, 2019, pp. 58–63. doi: 10.1007/978-3-030-33702-5_5.

[24] F. Auer, V. Lenarduzzi, M. Felderer, and D. Taibi, "From monolithic systems to Microservices: An assessment framework," *Inf. Softw. Technol.*, vol. 137, May 2021, doi: 10.1016/j.infsof.2021.106600.

[25] T. Kinoshita and H. Kanuka, "Automated Microservice Decomposition Method as Multi-Objective Optimization," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2022, pp. 112–115. doi: 10.1109/ICSA-C54293.2022.00028.

[26] A. Christoforou, L. Odysseos, and A. S. Andreou, "Migration of software components to microservices: Matching and synthesis," in *ENASE 2019 - Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, E. Damiani, G. Spanoudakis, and L. Maciaszek, Eds., SCITEPRESS - Science and Technology Publications, 2019, pp. 134–146. doi: 10.5220/0007732101340146.

[27] A. Christoforou, A. S. Andreou, M. Garriga, and L. Baresi, "Adopting microservice architecture: A decision support model based on genetically evolved multi-layer FCM," *Appl. Soft Comput.*, vol. 114, p. 108066, May 2022, doi: 10.1016/j.asoc.2021.108066.

[28] K. Sellami, A. Ouni, M. A. Saied, S. Bouktif, and M. W. Mkaouer, "Improving microservices extraction using evolutionary search," *Inf. Softw. Technol.*, vol. 151, p. 106996, 2022, doi: https://doi.org/10.1016/j.infsof.2022.106996.

[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: 10.1109/4235.996017.

[30] J. D. Schaffer, "Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms," Vanderbilt University, 1985.

[31] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, 2014, doi: 10.1109/TEVC.2013.2281535.

[32] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," 2001. doi: https://doi.org/10.3929/ethz-a-004284029.

[33] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, May 2007, doi: 10.1109/TEVC.2007.892759.

[34] E. Zitzler and S. Künzli, "Indicator-Based Selection in Multiobjective Search," in *Parallel Problem Solving from Nature - PPSN VIII*, E. K. and L. J. A. and S. J. and M.-G. J. J. and B. J. A. and R. J. E. and T. P. and K. A. and S. H.-P. Y. X. and Burke, Ed., Springer Berlin Heidelberg, 2004, pp. 832–842. doi: 10.1007/978-3-540-30217-9_84.

[35] K.-H. Chow, U. Deshpande, V. Deenadayalan, S. Seshadri, and L. Liu, "Atlas: Hybrid Cloud Migration Advisor for Interactive Microservices," in *EuroSys 2024 - Proceedings of the 2024 European Conference on Computer Systems*, ACM, May 2024, pp. 870–887. doi: 10.1145/3627703.3629587.

[36] S. Izadpanah, A. Rasoolzadegan, S. Abrishami, and A. Mousavi, "Improving Microservices Identification for Migration to Cloud-Native Applications," *IEEE Trans. Serv. Comput.*, pp. 1–15, Apr. 2026, doi: 10.1109/TSC.2026.3658420.

[37] A. Mwotil, B. Kanagwa, and E. Bainomugisha, "Yonga: An Adaptive Metaheuristic-Based Microservice Placement Approach for Low-Resource Distributed Systems," *IEEE Access*, vol. 14, pp. 6647–6663, Apr. 2026, doi: 10.1109/ACCESS.2026.3653120.

[38] W. D. F. Mendonça, W. K. G. Assunção, L. V Estanislau, S. R. Vergilio, and A. Garcia, "Towards a Microservices-Based Product Line with Multi-Objective Evolutionary Algorithms," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, Jul. 2020, pp. 1–8. doi: 10.1109/CEC48606.2020.9185776.

[39] W. K. G. Assunção *et al.*, "Analysis of a many-objective optimization approach for identifying microservices from legacy systems," *Empir. Softw. Eng.*, vol. 27, no. 2, Mar. 2022, doi: 10.1007/S10664-021-10049-7.

[40] M. Abdellatif *et al.*, "A taxonomy of service identification approaches for legacy software systems modernization," *Journal of Systems and Software*, vol. 173, May 2021, doi: 10.1016/j.jss.2020.110868.

[41] G. Filippone, M. Autili, F. Rossi, and M. Tivoli, "Migration of Monoliths through the Synthesis of Microservices using Combinatorial Optimization," in *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Oct. 2021, pp. 144–147. doi: 10.1109/ISSREW53611.2021.00056.

[42] R. Yedida, R. Krishna, A. Kalia, T. Menzies, J. Xiao, and M. Vukovic, "Lessons learned from hyper-parameter tuning for microservice candidate identification," in *Proceedings - 2021 36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021*, IEEE, May 2021, pp. 1141–1145. doi: 10.1109/ASE51524.2021.9678704.

[43] R. Yedida, R. Krishna, A. Kalia, T. Menzies, J. Xiao, and M. Vukovic, "An expert system for redesigning software for cloud applications," *Expert Syst. Appl.*, vol. 219, p. 119673, May 2023, doi: 10.1016/j.eswa.2023.119673.

[44] M. Camilli, C. Colarusso, B. Russo, and E. Zimeo, "Domain Metric Driven Decomposition of Data-Intensive Applications," in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Oct. 2020, pp. 189–196. doi: 10.1109/ISSREW51248.2020.00071.

[45] M. Camilli, C. Colarusso, B. Russo, and E. Zimeo, "Actor-Driven Decomposition of Microservices through Multi-level Scalability Assessment," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 5, May 2023, doi: 10.1145/3583563.

[46] R. Szlapczynski and J. Szlapczynska, "W-dominance: Tradeoff-inspired dominance relation for preference-based evolutionary multi-objective optimization," *Swarm Evol. Comput.*, vol. 63, p. 100866, Jun. 2021, doi: 10.1016/J.SWEVO.2021.100866.