# i-toMicroservices: Empowering Architects in the Interactive Modernization of Monolithic Systems

João Castanho
João Gomes
State University of Maringá
Maringá, Brazil
joao.staub42@gmail.com
joaoalt0502@gmail.com

Thelma E. Colanzi
Aline M. M. M. Amaral
State University of Maringá
Maringá, Brazil
thelma@din.uem.br
ammmamaral@uem.br

Wesley K. G. Assunção
NC State University
Raleigh, USA
wguezas@ncsu.edu

Alessandro Garcia
PUC-Rio
Rio de Janeiro, Brazil
afgarcia@inf.puc-rio.br

## Abstract

Companies have been migrating monolithic legacy systems to a microservice architecture to achieve better modularity, maintainability, and scalability. This process is particularly challenging when identifying microservices to decompose a monolithic architecture. To aid architects, several automated approaches have been proposed. However, recent studies have shown that modernizing monolithic systems should also integrate architects' preferences into the microservice decomposition process. As existing automated approaches do not fully cover this integration, we developed *i-toMicroservices*, a tool that provides full-fledged support for real-time, interactive refinement. By including the human in the loop directly in the search process, the tool enables refinements through new proposed operators, such as Split, Move, Join, and Freeze, that allow microservices to be adapted to the architect's preferences during the search process. This interactive capability provides powerful decision-support means that bridge the gap between metric optimization and expert domain knowledge, ultimately reducing human fatigue and delivering highly practical, domain-aligned microservice architectures for industrial migrations.

**Demo video:** https://doi.org/10.6084/m9.figshare.33189948

## Keywords

Microservice, Monolith decomposition, Search-based software engineering, Interactive optimization, Human in the loop, NSGA-III

## 1 Introduction

Companies have been migrating their legacy systems to microservices to overcome challenges inherent in monolithic architectures, such as scalability limitations, operational inefficiency, and difficulties in adopting new technologies [18]. In a microservices architecture, the system is decomposed into small and autonomous services with scopes limited to well-defined business functions [33]. However, the transition from a monolith legacy system to microservices is a highly complex process, often performed manually and strictly dependent on the intuition of software architects [39]. In the field of search-based software engineering, several approaches [4, 23–26, 30, 34, 36, 38, 41] offer automated solutions to mitigate this issue by using a set of guiding metrics and treating microservice decomposition as a combinatorial optimization problem [21]. A prominent example of this trend is the *toMicroservices* approach [4], which uses the NSGA-III many-objective evolutionary algorithm to identify optimized partitions based on metrics for coupling, cohesion, feature modularity, resuse, and network overhead.

Despite the effectiveness of toMicroservices in generating metrically robust solutions detailed at the method level [4], its original operation occurs under a static, non-interactive perspective [40]. This approach represents a common limitation among existing tools, which often operate in isolation as "black boxes", preventing the iterative refinements necessary in real-world scenarios [6, 40]. Other search-based approaches for microservice decomposition [1, 23–26, 30, 34, 36, 38, 41] have similar characteristics, ranging from none to only limited support for iterative solution refinements (Section 3).

To bridge this gap, this work presents *i-toMicroservices*, a tool specifically designed to serve software architects involved in large-scale migrations. The tool seeks to address the lack of acceptability of purely automated architectures [6] by transforming the optimization process into a human-in-the-loop system. To achieve this, *i-toMicroservices* transforms the traditional search process adopted in its precursor toMicroservices approach into an interactive optimization platform [27]. By inserting the decision maker directly into the evolutionary cycle, the tool allows the specialist to act as an "enricher" who guides the search algorithm. The alignment with business objectives is driven entirely by the architect's subjective knowledge, who uses their expertise to interpret the solutions and steer the search toward regions of the solution space that technical metrics alone cannot capture.

The interaction with the tool is mediated by an interface, in which the user applies refinement operators to direct the search algorithm's trajectory. Our tool includes new operators: freezing, slipt or join of components, and movement of methods. To ensure scalability and mitigate user fatigue, an intelligent clustering propagation mechanism replicates decisions across similar solutions in the population, ultimately increasing the acceptability of the generated microservices. Given the *i-toMicroservices* functionalities, including its interactive capabilities to include domain knowledge, decision makers are provided with a decision-support environment to conduct the migration from a monolithic to a microservice architecture, thereby reducing human fatigue and delivering practical, domain-aligned microservices for industrial migrations. Hence, the core contribution of this work does not lie in the underlying optimization algorithm, which builds upon the established NSGA-III engine, but in the interactive framework, the refinement operators and the clustering-based propagation mechanism, that operationalizes human-in-the-loop decision-making on top of this algorithm.

## 2 Background

This section provides the theoretical foundations underlying the automated and interactive decomposition of software architectures.

**Search-Based Software Engineering**. The Search-Based Software Engineering (SBSE) field reformulates software development and maintenance activities — such as architectural design and refactoring — into combinatorial optimization problems [19]. Instead of computing a single solution through exhaustive methods, SBSE employs metaheuristic algorithms to explore vast and complex search spaces, looking for satisfactory solutions within viable computational limits [21]. The core of this approach relies on a fitness function, which quantifies the quality of candidate architectures by consolidating conflicting software metrics [32].

**The NSGA-III and the Pareto Frontier.** To handle the intrinsically multi-objective nature of monolith decomposition, the toMicroservices approach relies on the Non-dominated Sorting Genetic Algorithm III (NSGA-III) [12]. NSGA-III is an evolutionary algorithm designed for optimization problems with multiple conflicting criteria. It operates through an iterative process that evolves a population of candidate solutions over generations using selection, crossover, and mutation operators [13]. Its main innovation is a selection mechanism based on well-distributed reference points and adaptive normalization, ensuring population diversity and uniform convergence toward the optimal frontier [10].

**Solution Representation.** In this search-based optimization context, candidate architectures (or individuals) within the population must be modeled appropriately to allow algorithm manipulation [20]. In toMicroservices, the system topology is represented using a graph model, where each vertex corresponds to a software method associated with a specific feature, and edges denote method-level communications (encompassing static calls, dynamic calls, and the volume of data transferred) [31]. During the evolutionary search, the genetic operators dynamically alter these method clusterings to generate new architectural candidate solutions.

**Interactive Optimization.** The decision maker (DM) is the agent responsible for the selection of the solution within the set of compromises generated by the search. The success of multi-objective approaches in SBSE critically depends on the alignment between the search process and the DM's preferences. As demonstrated by the fundamental goal of multi-objective optimization algorithms, the aim is not simply to generate a set of metrically optimal solutions, but rather to produce a set consistent with the DM's priorities [29]. In the context of decomposing legacy systems into microservices, the DM's preferences can vary by scenario, and the inappropriate selection of quality indicators without considering these preferences can lead to results that do not meet practical needs. As illustrated by a set of solutions that can be evaluated as technically superior, but are rejected by the DM for not aligning with their subjective criteria and architectural priorities [29].

The ways of incorporating DM preferences can be organized into three main approaches: (i) *a priori*: preferences are established before the search starts to guide the algorithm from its initialization; (ii) *a posteriori*: preferences are considered after the search is executed, leaving it to the DM to select the final solution within a set of non-dominated solutions; and (iii) *interactive*: inserts the decision maker into the search cycle, allowing preferences to be refined or adjusted during the algorithm's execution. This continuous interaction, or human-in-the-loop, aims to align the search with the DM intuition and knowledge [5, 11, 22]. The application of the human-in-the-loop technique in SBSE is particularly beneficial for complex problems that depend on subjective criteria and tacit knowledge. The engineer acts as an "enricher", selecting solutions that align with desired design principles, which improves search efficiency by directing it to promising regions and ensures that the decomposition is pragmatically viable [35].

In this context, the main contribution of *i-toMicroservices* is providing a fully functional environment that operationalizes these interactive benefits for software architectural modernization. By offering an intuitive interface, the tool empowers DM to easily act as enrichers during the search process, eliminating the typical "black box" limitation of traditional metaheuristics.

Recent advances explore the use of Large Language Models (LLMs) and multi-agent systems to mitigate manual effort in legacy modernization. Tools such as Mono2Sls [9] propose fully automated multi-agent pipelines to generate microservice code without human intervention. Along the same autonomous line, there are predictive approaches based on graphs and embeddings, such as MicroDec [2] and MonoEmbed [37]. Despite the potential of LLMs to capture domain semantics, we opted not to rely on them in *i-toMicroservices* due to their performance degradation in high-dimensionality problems compared to evolutionary algorithms. Furthermore, the probabilistic nature and lack of reproducibility of LLMs introduce risks of architectural hallucinations, hindering the mathematical consistency required for the interaction of DMs.

## 3 Related Tools

The literature presents various strategies for microservice identification that use optimization algorithms, varying in levels of automation and granularity. One approach based on combinatorial optimization [16] uses graphs to group methods and classes, automatically generating microservice code and APIs. This approach focuses on fine granularity but operates fully automatically, without allowing user intervention during decomposition. Differently, FoSCI (Functionality-oriented Service Candidate Identification) [23] uses execution traces to create "functional atoms" via the Jaccard Coefficient, and later groups them using a modified NSGA-II algorithm to define service candidates.

MSExtractor [36] addresses the problem using the IBEA evolutionary algorithm, analyzing source code at the class level and distinguishing internal classes from interface classes to refine the coupling measurement. MSExtractor offers an initial interaction, allowing the selection of solutions via a graphical interface after execution. The SAMIM approach [24] shifts the focus to the business level, using the MOEA/D [28]evolutionary algorithm, to partition activities defined in process models instead of code elements.

The evolution of human interaction is observed in the works of Kinoshita and Kanuka [25, 26]. Initially, they proposed a two-step method that uses reference lines to select Pareto-optimal candidates, aiming to reduce the architect's cognitive load. Subsequently, this method evolved to integrate NSGA-III, allowing the specialist not only to select but also to manually adjust the solution, bringing the final result closer to the DM's strategic perspective.

Other solutions integrate domain-specific criteria, such as So4MoD [30], which employs NSGA-II [14] to balance modularity with system security. Following a mathematical optimization

approach, Pangaea [38] and its successor, Cromlech [34], employ Mixed Integer Linear Programming (MILP) [17] to allocate data entities and operations at a high architectural level, allowing iterative user adjustments and cost/structural analysis via technology-agnostic models. Finally, addressing the data tier, the BLDD method targets database fragmentation by combining NSGA-II with spectral clustering, though it still requires human validation to handle strict relational constraints [41].

In contrast to the mentioned approaches, *i-toMicroservices* distinguishes itself by integrating the DM directly into the optimization cycle of NSGA-III. While tools like MSExtractor limit interaction to a posteriori phase and the works of Kinoshita and Kanuka [25, 26] focus on specific manual adjustments, the proposed tool introduces an intelligent propagation mechanism via clustering. This differentiator allows the user's architectural decisions to be automatically replicated across similar solutions in the population. As a result, human fatigue is reduced while the search is guided toward regions of the solution space that align rigorous technical metrics with subjective domain knowledge.

# 4 The i-toMicroservices Tool

## 4.1 toMicroservices Foundation

The original toMicroservices approach [4] is the direct precursor and foundation of *i-toMicroservices*, serving as a semi-automated, search-based approach for identifying microservices in legacy systems. It distinguishes itself by simultaneously optimizing five essential criteria: coupling, cohesion, feature modularization, network overhead, and reuse [3]. Different from many approaches that operate only at the class or file level, the analysis is performed at the method level (Section 3). This level enables fine-grained granularity and yields architectures that developers can adopt as a starting point for modernization [4, 6–8].

**Representation, Input, and Output**. The approach models the system architecture as a graph, with each vertex representing a method associated with a feature [3]. Edges represent communication between methods via a triple $e = (sc, dc, ds)$, where $sc$ are static calls, $dc$ are dynamic calls, and $ds$ is the size of the data transferred. The approach requires three inputs: the list of features, the initial monolith architecture, and the desired number of microservices. The process follows a multi-objective optimization process (using NSGA-III) that results in a set of Pareto-optimal architectures. Currently, the DM's (*Decision Maker* - DM) interaction occurs *a priori*, during configuration, and *a posteriori*, during the inspection of output files to validate metrics and the suggested topology. Unlike this static workflow, *i-toMicroservices* inserts the DM directly inside the evolutionary loop, enabling real-time refinement of candidate architectures through the Split, Move, Join, and Freeze operators (Section 4.4), instead of restricting the architect's participation to the boundaries of the process.

**Objective Functions**. The toMicroservices approach identifies microservice candidates by simultaneously optimizing five software metrics that reflect both technical and business perspectives [3]. **Cohesion**: Measures the internal strength of a microservice by analyzing the interrelation between its methods. The goal is to maximize the frequency of internal calls, ensuring that each service has a single, well-defined purpose. **Coupling**: Quantifies the external dependencies of a microservice by summing the static calls to methods located outside its boundaries. To ensure service autonomy and independence, the approach seeks to minimize these external connections. **Network Overhead**: Estimates the communication cost between services, considering elements such as HTTP headers and data volume. The optimization process minimizes the maximum parameter traffic to reduce latency and infrastructure costs. **Reuse**: Evaluates whether a microservice is invoked by multiple components or end-users. The tool identifies potentially shared services and aims to maximize the overall architecture's reusability. **Feature Modularization**: Groups methods based on their respective business functions. It measures the dominance of a main feature within a service to ensure the decomposition remains aligned with the system's functional domains and to maximize this alignment.

## 4.2 Version Interactive: Functionalities and Users

*i-toMicroservices* is designed to support software architects and systems engineers in the complex process of modernizing monolithic architectures. The tool's primary goal is to allow these specialists to identify microservice candidates in an assisted manner, combining the precision of optimization algorithms with strategic human knowledge. To address this, the tool provides a collaborative environment where the user can: **(1) Configure and Control**: define the target number of microservices and the timing of the interventions. **(2) Evaluate and Compare**: use a dashboard to compare different architectural candidates based on metrics trade-offs between coupling, cohesion, network overhead, and functionality. **(3) Intervene and Refine**: act directly on the architectural graph to apply domain knowledge, such as merging services that share a database or splitting services that grew too large, effectively guiding the search process toward a feasible and maintainable design.

By facilitating this human-in-the-loop approach, the tool seeks to reduce the gap between academic optimization and industrial application. It serves as a decision support platform that attempts to minimize the risks and manual effort associated with large-scale architectural modernization. For the user, the tool offers an environment where it is possible to configure migration scenarios, visualize architectural alternatives through graphs, and directly intervene in the composition of services. Unlike purely automated approaches, *i-toMicroservices* allows the DM to apply fine-tuning actions, such as moving methods between microservices, splitting or joining components, and freezing microservices. This ensures that the final result respects business rules and technical constraints that automated metrics cannot capture in isolation.

## 4.3 Architecture

The architecture of *i-toMicroservices* follows a client-server model, as illustrated in Figure 1. The system is designed to support the processing of evolutionary algorithms while maintaining a responsive user interface. This architectural choice decouples the computationally intensive evolutionary process, executed in the back-end, from a lightweight and reactive front-end, allowing the search engine to be paused, queried, and resumed in real time with progress in the user interface (Section 4.4). The tool's back-end serves as the processing core and the intelligence hub of the approach. All SBSE logic

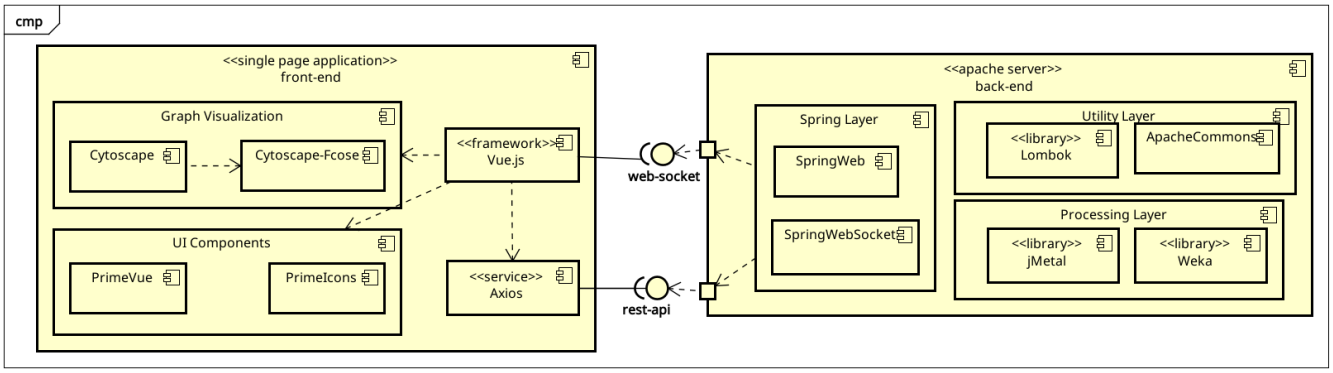



**Figure 1: Architecture of i-toMicroservices**

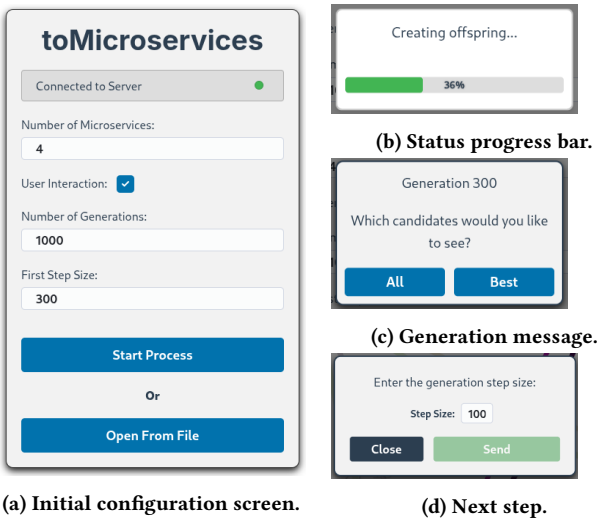
is contained within this module, which integrates the jMetal framework[1] as the primary engine for NSGA-III. To enable interactivity, the back-end uses the Weka library, specifically the K-means++ algorithm, which enables clustering of solutions with similar metric profiles to propagate the DM interaction. Additionally, libraries such as Apache Commons and Lombok are used to ensure robust data handling and reduce code verbosity, thereby facilitating the tool's maintenance and evolution. The choice of jMetal as the optimization engine is motivated by its native, well-established support for NSGA-III and its extensibility for encoding custom solution representations. Weka's K-means++ implementation was adopted for its efficiency in clustering high-dimensional metric profiles, which is essential to sustain the real-time propagation mechanism.

The front-end was developed as a Single Page Application (SPA) using the Vue.js framework, focusing on an intuitive interface for the specialist. The visualization of decomposition proposals is handled using the Cytoscape library with the ForceAtlas2 layout, which organizes microservices and their interdependencies into readable, interactive graphs. Communication between modules occurs in a hybrid, decoupled manner: while the REST protocol, managed by the Axios library, is used to send structured commands and configurations, the WebSockets protocol (via SpringWebSocket) establishes a real-time bidirectional channel. This communication infrastructure is the technical differentiator that allows the search engine to pause execution and expose the algorithm's state instantaneously upon user command, providing the DM visibility into the steps the algorithm is executing.

## 4.4 Usage and New Interactive Operators

The home screen (Figure 2a) concentrates the basic execution parameters. At this stage, the user defines the number of microservices and enables the interaction mode, allowing the automatic flow to be interrupted at strategic points for analysis and refinement. Figure 2b presents a progress bar to clarify the algorithm for the DM. Once the generation process is completed, the tool directs the user to the candidate selection stage (Figure 2c). Then, the system displays the current generation and allows the user to choose between viewing the best-candidate solution and the complete list of generated solutions. This choice enables the user to perform a detailed analysis of

[1]https://github.com/jmetal/jmetal




**(a) Initial configuration screen.**

**(b) Status progress bar.**

**(c) Generation message.**

**(d) Next step.**

**Figure 2: Initial tool screens.**

the solution or proceed along the path automatically established by the genetic algorithm's evaluation and scoring mechanism. The interactive workflow is based on a scheduled interruption cycle. After the initial configuration, the search engine executes autonomously for a predetermined number of generations (step size). When this milestone is reached, the evolution is paused, allowing the Decision Maker (DM) to inspect the current population and inject subjective knowledge before resuming the process.

The candidate solution listing (Figure 3) provides a set of metrics intended to facilitate the comparison and selection of architecture alternatives, including *coupling*, *cohesion*, *overhead*, *functionality*, and *reuse* for DM. In addition, the number of microservices associated with the evaluated candidate is displayed, along with a visual representation of the candidate that achieved the best score within the population. After selecting a candidate, the tool presents it as an architectural graph (Figure 4), showing the proposed decomposition and the services' dependencies. Microservices are represented as nodes, while edges denote method-level interactions between these microservices. The tool visually encodes the number of methods associated with each microservice through variations in node size,

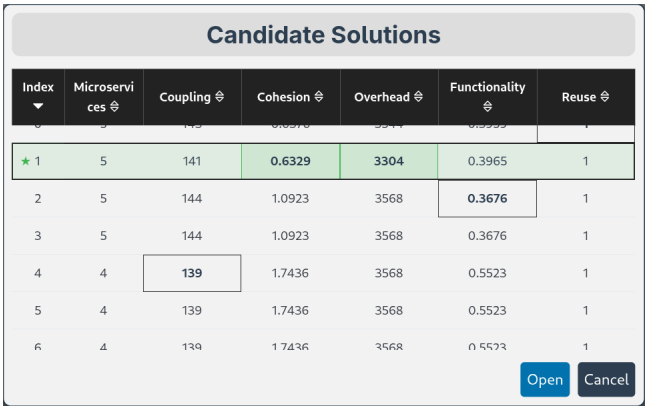



**Figure 3: List of candidate solutions.**

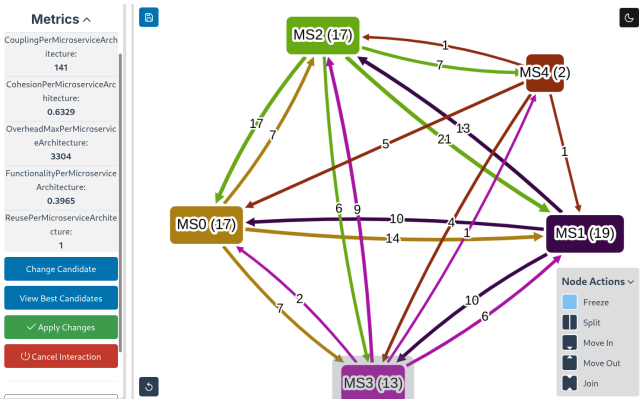



**Figure 4: Solution representation in graph format.**

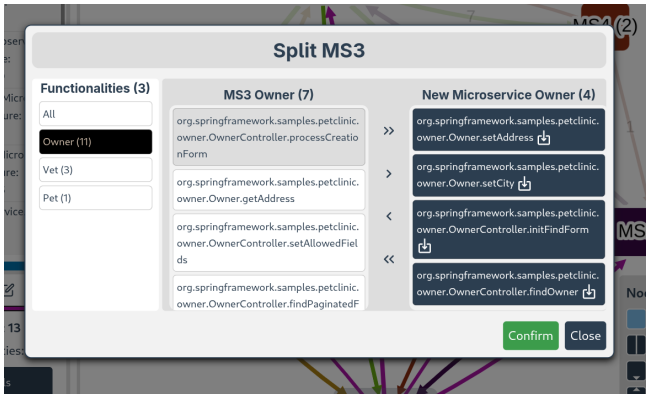



**Figure 5: Execution of the *Split* operator.**

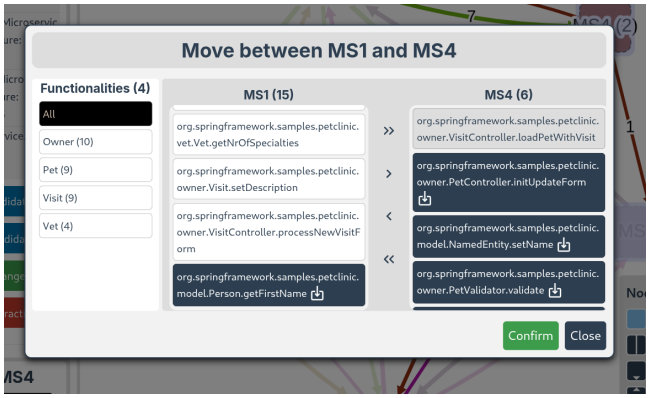



**Figure 6: Execution of the *Move* operator.**

applying a similar visual encoding to edges as well. The interface on the left of the screen provides contextual actions applicable to the graph. It allows the user to modify the solution and influence subsequent algorithm generations. It also enables the user to open a detailed view of each selected microservice or edge, including a complete listing of the methods associated with the corresponding element. User intervention occurs by selecting a representative solution and applying structural changes. These modifications are not restricted to a single individual; through an intelligent propagation mechanism, the tool identifies similar solutions in the population and replicates the architect's intent across them. This ensures that DM refined traits are prioritized in next evolutionary generations.

Notice that the presented metric values are the objective function values computed by NSGA-III to rank and select solutions during the evolutionary process (Section 4.1). Therefore, the values shown to the DM at each interruption point are as robust and reliable as the metrics used internally by the search algorithm, ensuring that decisions made during the interactive process are grounded in accurate architectural measurements.

*4.4.1 Interactive Execution of Split and Move Operators.* The *Split* operator allows separating methods from a microservice to form a new unit (Figure 5), and redistribution of methods to create a new microservice. The interface presents the associated features and the distribution of methods, favoring a transparent decision about the solution's fragmentation. The *Move* operator enables the reallocation of methods between existing microservices (Figure 6) and the controlled repositioning of responsibilities between source and target services. This operation is useful for fine-tuning responsibility while preserving the general structure and modifying the internal distribution of components. Finally, after the changes, the tool requests consolidation of the intervention (Figure 2d). This step formalizes the knowledge injected by the DM, allowing evolution to be resumed based on a population qualified by DM.

*4.4.2 Interactive Execution of Join and Freeze Operators.* The **Join** operator allows the merging of two microservices into a single unit when the DM identifies they share a common business domain not captured by the automated metrics. The **Freeze** operator enables the user to lock a specific microservice that is already considered ideal. Once frozen, the methods within this service are protected from mutations, preserving the architect's validated design. Together, these features show that the tool organizes a workflow where configuration, comparison, inspection, and manipulation coexist, increasing control over the quality of the decompositions.

## 5 Two-Phase Evaluation

The consolidation of *i-toMicroservices* was driven by a two-phase integrative evaluation process that bridges empirical industrial findings with active tool usability validation.

**Phase 1: Reinforcing User Interaction Needs.** The 1st phase comprised an industrial on-site study, conducted using the original, fully automated version of the tool, toMicroservices, which provides no interaction support. Our goal was to pinpoint tool limitations, while eliciting whether and which kind of human-in-the-loop support was needed for *i-toMicroservices*. Software architects evaluated decompositions of a large-scale system comprising over 15,000 classes and 2,300 database tables [15]. Although the multi-objective engine proved to be metrically robust, generating high-quality technical frontiers, the study revealed that purely automated solutions are rarely adopted in practice without extensive manual intervention [15, 40]. Architects frequently rejected or heavily modified the recommended architectures due to: (i) not feeling they had the ownership of the generated architectures, and (ii) misalignment with certain subjective goals [40].

There were various examples of design goals that could only emerge along the process of generating the architectures. First, developers wanted to refine certain medium-sized components and interfaces into more fine grained ones as they found out that, while discussing, a pair of developers would prefer to individually work in separate microservices. This observation justified the following features of *i-toMicroservices*: (i) the need for Split and Move operators (Section 4.4.1), (ii) the configuration of the interaction mode (Section 4.4), (iii) the option of choosing either the best-candidate solution or the complete list (Section 4.4), and (iv) the number of expected microservices (Section 4.2). Second, the identification of opportunities for offering certain microservices' interfaces depended on architects iteratively reflecting upon business clients along the inspection of the current population (Section 4.4). Whenever they concluded that emerging interfaces could compose a single microservice, fully aligned with a business goal, they could opt for using Joing and Freeze operators (Section 4.4.2). Third, the architects mentioned they prefer to reflect upon the measures – e.g., coupling and network overhead – along the refinement process rather than only rigidly observing the resulting measures at the final outcomes, which is a limitation of toMicroservices and other tools (Section 3). These findings revealed a need to keep the architect directly in the evolutionary cycle to enable real-time structural adjustments.

To address the limitations clearly identified in the industrial study, the current version of the tool was designed to transform the static microservice identification process into an interactive experience. The *i-toMicroservices* directly materializes the requirements elicited from the industry, mentioned above, by introducing specific refinement operators, such as Split, Move, Join, and Freeze, allowing the architect to act as an active decision-maker and apply domain knowledge to correct the search trajectory during the algorithm's execution. Furthermore, to minimize user effort and avoid the cognitive fatigue associated with manual architectural adjustments, the tool employs an intelligent clustering-based propagation mechanism (Section 4.4). This component helps users amplify their strategic choices by automatically applying a single design adjustment across similar solutions in the population. Therefore, the architectural evolution presented in this paper directly answers empirical industry needs, establishing a cooperative decision-support platform ready for real-world deployment.

**Phase 2: User Experience.** To preliminarily assess the usability and interaction dynamics of the tool, the authors conducted exploratory testing sessions acting as domain experts. During the simulations, the graph-based interface enabled a fluid and immediate inspection of method-level interdependencies, facilitating the visual identification of architectural anomalies. The interactive operators (such as Split and Move) proved to be intuitive (Sections 4.4.1 and Section 4.4.2), allowing fine-grained adjustments directly onto the components without requiring manual code workarounds. The most prominent highlight observed during the evaluation was the intelligent clustering-based propagation mechanism (Section 4.4): upon applying a single design choice, the algorithm successfully replicated the architect's intent across semantically similar solutions within the population. This behavior drastically reduced repetitive fine-tuning effort, confirming that the subjective domain knowledge could guide the NSGA-III convergence trajectory in a transparent and highly responsive manner. Ultimately, these practical capabilities confirm that *i-toMicroservices* is specifically tailored for software architects tasked with managing complexity and executing large-scale architectural migrations.

## 6 Conclusion and Future Work

This work presented *i-toMicroservices*, shifting the perspective from static multi-solution generation to an interactive process. By combining a graphical interface with an intelligent propagation mechanism, the tool connects technical metric rigor with the specialist's tacit knowledge. The interactive operators, such as Split, Move, Join, and Freeze, allow fine-tuning of the balance among coupling, cohesion, and feature modularization. Future work will focus on evaluating the tool's impact on reducing architects' cognitive load during large-scale migrations. Additionally, new studies will assess whether the DM's preferences were effectively absorbed by the algorithm, using smaller-scale open-source systems for controlled validation. We also plan to explore the inclusion of domain-similarity objectives and to add machine learning techniques, such as LLM-based agents, to perform automated operations based on learned DM preferences.

## Artifact Availability

The source code of *i-toMicroservices*, along with installation guides and evaluation datasets, is available on Figshare under the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) license. Access is available at: https://figshare.com/s/70a1be8007284c54aa96.

## Acknowledgments

The work is supported by CAPES (Code 001), CNPq (404027/2023-7, 306774/2025-9), Fundação Araucária (Grant No. 792/2024), and FAPERJ (211.033/2019, 200.510/2023, 310391/2025-3). Additionally, the Gemini was used in assisting a initial text translation to English.

## References


[1] Yalemisew Abgaz, Andrew McCarren, Peter Elger, David Solan, Neil Lapuz, Marin Bivol, Glenn Jackson, Murat Yilmaz, Jim Buckley, and Paul Clarke. 2023. Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review. *IEEE Transactions on Software Engineering* 49, 8 (2023), 4213–4242. doi:10.1109/TSE.2023.3287297

[2] Ahmed Saeed Alsayed, Hoa Khanh Dam, and Chau Nguyen. 2024. MicroDec: Leveraging Large Language Models for Microservice Decomposition. *Journal of LaTeX* (sept 2024).

[3] Wesley KG Assunção, Thelma Elita Colanzi, Luiz Carvalho, Alessandro Garcia, Juliana Alves Pereira, Maria Julia de Lima, and Carlos Lucena. 2022. Analysis of a many-objective optimization approach for identifying microservices from legacy systems. *Empirical Software Engineering* 27, 2 (2022), 51.

[4] Wesley K. G. Assunção, Thelma Elita Colanzi, Luiz Carvalho, Juliana Alves Pereira, Alessandro Garcia, Maria Julia de Lima, and Carlos Lucena. 2021. A Multi-Criteria Strategy for Redesigning Legacy Features as Microservices: An Industrial Case Study. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 377–387. doi:10.1109/SANER50967.2021.00042

[5] Jürgen Branke. 2008. *Multiobjective optimization: Interactive and evolutionary approaches*. Vol. 5252. Springer Science & Business Media.

[6] Luiz Carvalho, Thelma Elita Colanzi, Wesley K. G. Assunção, Alessandro Garcia, Juliana Alves Pereira, Marcos Kalinowski, Rafael Maiani de Mello, Maria Julia de Lima, and Carlos Lucena. 2024. On the Usefulness of Automatically Generated Microservice Architectures. *IEEE Transactions on Software Engineering* 50, 3 (2024), 651–667. doi:10.1109/TSE.2024.3361209

[7] Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K. G. Assunção, Maria Julia Lima, Baldoino Fonseca, Mârcio Ribeiro, and Carlos Lucena. 2020. Search-based many-criteria identification of microservices from legacy systems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion* (Cancún, Mexico) *(GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 305–306. doi:10.1145/3377929.3390030

[8] Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K. G. Assunção, Juliana Alves Pereira, Baldoino Fonseca, Márcio Ribeiro, Maria Julia de Lima, and Carlos Lucena. 2020. On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 569–580. doi:10.1109/ICSME46990.2020.00060

[9] Xingyan Chen, Yuxin Su, Zishan Su, Yang Yu, and Zibin Zheng. 2026. Mono2Sls: Automated Monolith-to-Serverless Migration via Multi-Stage Pipeline with Static Analysis. *arXiv preprint arXiv:2604.24550* (apr 2026). arXiv:2604.24550 [cs.SE]

[10] Kalyanmoy Deb. 2011. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 3–34.

[11] Kalyanmoy Deb. 2012. Advances in evolutionary multi-objective optimization. In *International Symposium on Search Based Software Engineering*. Springer, 1–26.

[12] Kalyanmoy Deb and Himanshu Jain. 2013. 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* 18, 4 (2013), 577–601.

[13] Kalyanmoy Deb and Himanshu Jain. 2014. 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* 18, 4 (2014), 577–601. doi:10.1109/TEVC.2013.2281535

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. doi:10.1109/4235.996017

[15] Fernando S. Felizardo, Vinicius L. Nogueira, João V. S. Castanho, Thelma Elita Colanzi, Aline M. M. M. Amaral, and Wesley K. G. and Assunção. 2026. Microservices by Optimization and Experience: Lessons from a Large-Scale Industrial Migration. In *2026 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) - Industrial Track*.

[16] Gianluca Filippone, Marco Autili, Fabrizio Rossi, and Massimo Tivoli. 2021. Migration of monoliths through the synthesis of microservices using combinatorial optimization. In *2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 144–147.

[17] Christodoulos A Floudas and Xiaoxia Lin. 2005. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research* 139, 1 (2005), 131–162.

[18] Susan J. Fowler. 2017. *Microsserviços Prontos para a Produção* (1 ed.). Novatec Editora, São Paulo.

[19] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang. 2014. Search Based Software Engineering for Software Product Line Engineering: A Survey and Directions for Future Work. In *18th International Software Product Line Conference - Volume 1*. Association for Computing Machinery, Florence, Italy, 5–18.

[20] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Comput. Surv.* 45, 1, Article 11 (Dec. 2012), 61 pages. doi:10.1145/2379776.2379787

[21] Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. 2012. *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–59. doi:10.1007/978-3-642-25231-0_1

[22] Jan Hettenhausen, Andrew Lewis, Marcus Randall, and Timoleon Kipouros. 2013. Interactive multi-objective particle swarm optimisation using decision space interaction. In *2013 IEEE Congress on Evolutionary Computation*. IEEE, 3411–3418.

[23] Wuxia Jin, Ting Liu, Yuanfang Cai, Rick Kazman, Ran Mo, and Qinghua Zheng. 2021. Service Candidate Identification from Monolithic Systems Based on Execution Traces. *IEEE Transactions on Software Engineering* 47, 5 (2021), 987–1007. doi:10.1109/TSE.2019.2910531

[24] Sedigheh Khoshnevis. 2023. A search-based identification of variable microservices for enterprise SaaS. *Frontiers of Computer Science* 17, 3 (2023), 173208.

[25] Takahiro Kinoshita and Hideyuki Kanuka. 2022. Automated microservice decomposition method as multi-objective optimization. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 112–115.

[26] Takahiro Kinoshita and Hideyuki Kanuka. 2024. Enhancing automated microservice decomposition via multi-objective optimization. *IEEE Access* 12 (2024), 55697–55710.

[27] Giomara Lárraga, Bhupinder Singh Saini, and Kaisa Miettinen. 2023. Incorporating Preference Information Interactively innbsp;NSGA-III bynbsp;thenbsp;Adaptation ofnbsp;Reference Vectors. In *Evolutionary Multi-Criterion Optimization: 12th International Conference, EMO 2023, Leiden, The Netherlands, March 20–24, 2023, Proceedings* (Leiden, The Netherlands). Springer-Verlag, Berlin, Heidelberg, 578–592. doi:10.1007/978-3-031-27250-9_41

[28] Hui Li and Qingfu Zhang. 2008. Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II. *IEEE transactions on evolutionary computation* 13, 2 (2008), 284–302.

[29] Miqing Li, Tao Chen, and Xin Yao. 2018. A critical review of: " a practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering": essay on quality indicator selection for SBSE. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*. 17–20.

[30] Xiaodong Liu, Zhikun Chen, Yu Qian, Chenxing Zhong, Huang Huang, Shanshan Li, and Dong Shao. 2024. Towards a security-optimized approach for the microservice-oriented decomposition. *Journal of Software: Evolution and Process* 36, 10 (2024), e2670.

[31] Brian S Mitchell and Spiros Mancoridis. 2008. On the evaluation of the bunch search-based software modularization algorithm. *Soft Computing* 12, 1 (2008), 77–93.

[32] Michael Mohan and Des Greer. 2018. A survey of search-based refactoring for software maintenance. *Journal of Software Engineering Research and Development* 6 (2018), 1–52.

[33] Sam Newman. 2015. *Building Microservices* (1st ed.). O'Reilly Media, Inc.

[34] Giovanni Quattrocchi, Davide Cocco, Simone Staffa, Alessandro Margara, and Gianpaolo Cugola. 2024. Cromlech: Semi-automated monolith decomposition into microservices. *IEEE Transactions on Services Computing* 17, 2 (2024), 466–481.

[35] Aurora Ramirez, Jose Raul Romero, and Christopher L Simons. 2018. A systematic review of interaction in search-based software engineering. *IEEE Transactions on Software Engineering* 45, 8 (2018), 760–781.

[36] Khaled Sellami, Ali Ouni, Mohamed Aymen Saied, Salah Bouktif, and Mohamed Wiem Mkaouer. 2022. Improving microservices extraction using evolutionary search. *Information and Software Technology* 151 (2022), 106996. doi:10.1016/j.infsof.2022.106996

[37] Khaled Sellami and Mohamed Aymen Saied. 2025. Contrastive Learning-Enhanced Large Language Models for Monolith-to-Microservice Decomposition. *arXiv preprint arXiv:2502.04604* (feb 2025). arXiv:2502.04604 [cs.SE]

[38] Simone Staffa, Giovanni Quattrocchi, Alessandro Margara, and Gianpaolo Cugola. 2021. Pangaea: Semi-automated monolith decomposition into microservices. In *International Conference on Service-Oriented Computing*. Springer, 830–838.

[39] Davide Taibi and Kari Systä. 2019. A Decomposition and Metric-Based Evaluation Framework for Microservices. *CoRR* abs/1908.08513 (2019). arXiv:1908.08513 http://arxiv.org/abs/1908.08513

[40] Yingying Wang, Sarah Bornais, and Julia Rubin. 2024. Microservice Decomposition Techniques: An Independent Tool Comparison. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (Sacramento, CA, USA) *(ASE '24)*. Association for Computing Machinery, New York, NY, USA, 1295–1307. doi:10.1145/3691620.3695504

[41] Peng Zhang, Weigang Li, Ruihan Gao, Henan Zhang, and Junsheng Wu. 2024. Research on database splitting methodology for microservicization of monolithic applications. In *2024 4th International Conference on Communication Technology and Information Technology (ICCTIT)*. IEEE, 562–566.