STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **x**, Month 202x, pp 0–16.
Published online in International Academic Press (www.IAPress.org)

Implementing Automatic Microservices Detection in Business Processes Using Association Rules

Mohamed Daoud [1], My Abdelouahed Sabri [2]

[1] *University Lyon 1, France*
[2] *Faculty of Sciences Dhar el Mahraz, University Sidi Mohamed Ben Abdellah,Fez, Morocco*

Abstract Migrating from monolithic to microservices architecture presents a significant challenge, particularly in large and complex systems. In a monolithic system, components are tightly coupled, making the process of identifying appropriate microservices boundaries difficult. This migration involves breaking down large applications into smaller, independent, and cohesive services that can operate with minimal dependencies. The need to ensure smooth transitions while maintaining system performance and preventing anomalies adds complexity to this transformation. Detecting patterns and potential anomalies in business processes during this migration is critical to avoiding bottlenecks, costly rework, and interruptions in service delivery. However, the process of identifying appropriate microservices is a significant challenge, which, if not addressed adequately, could hinder the effectiveness and benefits of transitioning to this architectural style.In this paper, we introduce an innovative method based on association rules to automate the identification of microservices within a business process. This technique leverages association analysis to uncover latent correlations among the attributes of various activities. Activities sharing similar attributes are then grouped into the same microservice categories. To validate and demonstrate the practicality of our method, we conduct a case study focusing on a bicycle rental system.

Keywords monolithic systems, microservices, Business process, association rules.

DOI: 10.19139/soic-2310-5070-2071

1. Introduction

The landscape of business today is marked by increasingly rapid shifts, driven by the relentless pursuit of competitiveness [16][15]. In this dynamic environment, the primary challenge for companies is to effectively manage their evolution, adapting their information systems and business processes to align with evolving work practices, emerging customer needs, and new regulatory requirements. Central to this adaptive capability is the business process, an organized amalgamation of activities, software, hardware, and human resources [11]. It serves as the backbone of any organization, with its economic success hinging on the agility of the business process in assimilating environmental changes. Traditionally, these business processes (BPs) are structured as monolithic systems. They are designed as single, indivisible units, with tightly interlinked and interdependent components [22], lacking the flexibility for easy maintenance or upgrades. This structure leads to inefficient and sluggish modifications and enhancements. In contrast to these monolithic systems, microservice-based architectures [21][6][13] have gained traction. They offer businesses the agility to swiftly adapt to new demands and circumvent protracted development cycles that span several years. This architectural paradigm decomposes a system into a suite of small, independent services, with each service managing its segment of the business process autonomously [1][8], and maintaining a distinct database. A key challenge in this domain is determining

*Correspondence to: Mohamed Daoud, Email:mtdaoud.info@gmail.com

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

the optimal granularity and cohesion of microservices, a consideration that is crucial both in initiating new projects and in managing the maintenance, evolution, and scaling of existing systems. Our paper addresses this challenge by proposing a methodology for identifying microservices based on their activity dependencies, focusing on control aspects and attribute correlations within the business process. We describe each activity using a collection of artifacts and attributes. Our goal is to achieve low coupling and high cohesion by structuring our system such that each microservice operates within its own domain, with exclusive operations and data. Essentially, our approach partitions the business process activities into distinct sets, where the operations within each microservice can only directly interact with their respective variables. To this end, we have adopted the association rules technique, a well-established method in data mining. This technique will be employed to identify significant correlations between different system attributes. Activities sharing data attributes will be grouped together, leading to a design where each microservice has a localized database. This approach effectively minimizes interactions between various system components, enhancing the overall efficiency and responsiveness of the business process.

2. Related work

Microservices have emerged as the preferred software architecture for business applications, surpassing traditional monolithic systems. This shift can be traced back to influential early adopters like Netflix and Amazon. The rationale behind this transition lies in the need to segregate software development teams and runtime components, fostering agile development methodologies and enhancing scalability.

In [23] explores a unique method for identifying microservices within business processes, leveraging the principles of association rules from data mining. This approach delves into understanding the intricate relationships between the attributes of various activities that make up a business process. By examining these relationships, the methodology aids in effectively classifying these activities into distinct microservices. The paper exemplifies this method through a case study of a bicycle rental system, providing a practical context. It thoroughly discusses the conceptual foundation of business processes, defining them as collections of related activities bound by logical relations and attribute associations. Finally, it outlines the algorithms used for measuring dependencies, generating frequent itemsets, and creating association rules, thereby providing a comprehensive framework for microservices identification in the realm of business processes.

In the realm of existing software, a significant number of monolithic applications are transitioning to a microservices architecture. This trend is driven by the benefits of microservices in terms of maintenance, scalability, and deployment flexibility.

A pertinent study in this field is presented in [9], where the authors discuss a methodology for transforming an integration platform, originally built on a Service-Oriented Architecture (SOA), into a microservices-oriented platform. This revamped platform addresses the limitations inherent in the original architecture, particularly concerning the processing of a large volume of messages and support for numerous new integrations. The microservices-based platform brings to the table enhanced flexibility and agility, especially in the maintenance, scalability, and deployment of software products.

Facing the challenges of maintaining and evolving substantial applications, enterprise developers seek efficient strategies. In this context, [10] details the work of Escobar et al., who propose a model-centric approach for analyzing and visualizing the structure of current applications along with the interdependencies between business and data capacities. This methodology facilitates the decomposition of Java EE (Enterprise Edition) applications into microservices. It employs a clustering technique to analyze data associated with each Enterprise Java Bean (EJB), resulting in informative diagrams that guide the decomposition process.

These evolving methodologies underscore the ongoing shift towards microservices, highlighting their advantages in handling complex, large-scale business applications.

In [1], authors have proposed a method of identifying microservices that decomposes a system using the clustering technique. To this end, they modeled a system as a set of business processes and they took into account two aspects of structural dependency and data dependency. In addition, the authors conducted a study to assess the effect of the process characteristics on the accuracy of identification approaches. Their approach is essentially

based on three steps: First, a relation TP which shows the structural dependence of activities within a business process. Then a relation TD is defined to show the dependency of the activities according to their used data objects and finally they aggregated these two relations to define the final relation T.
Amiri and al. were the first to work on identifying microservices from a set of BPs. However, they did not model the control dependency by taking into account the different types of logical connectors (Xor, or..).
Recently, the approach of Daoud and al.[8] was proposed to address the limitations of the approach of amiri et al. already mentioned in their work. The main goal of the approach is to automatically identify microservices based on two types of dependencies (control and data) using collaborative clustering. To do this Daoud et al. proposed formulas for calculating direct and indirect control dependencies as well as proposed two strategies for calculating data dependency. Then they used a collaborative clustering algorithm to automatically extract candidate microservices.
Although both works in [1][8] discuss control and data dependencies between activities, they did not take into account the correlation between attributes in order to reduce interactions and define a database per microservice.

There exists a significant body of work discussing the limitations of monolithic systems and how microservices can address these limitations [10]. Such systems are known for incurring significant development, maintenance, and evolution costs [24]. This section discusses works related to microservices identification, a critical step when migrating from monolithic applications to microservices-based applications.

In[1] propose a microservices decomposition methodology mapping functional requirements onto microservices, considering non-functional requirements like security and scalability. This approach expresses an application as a set of functional requirements, each corresponding to a system functionality, complemented by security requirements identified through misuse cases.

A clustering-based approach is discussed in [8], where structural and data dependencies between tasks from a set of Business Processes (BPs) are extracted. These dependencies are merged into a dependency matrix, then submitted to a classical clustering algorithm to identify candidate microservices.

In [3] propose a process to identify candidate microservices from a set of business rules implemented as stored procedures. This approach involves discovering stored procedures related to requirements, analyzing them to identify candidate microservices, and evaluating the source code of these candidates for refinement.

In [4] propose an approach using the semantic similarity of functionalities described in OpenAPI and a reference vocabulary. They identify necessary microservices as cohesive clusters of operations extracted from UML diagram classes, with semantic similarities based on the pre-computed DISCO database [18].

Li et al. [6] propose a data flow-driven approach using Data Flow Diagrams (DFDs) generated from business logic to describe data flow through business processes. These DFDs are transformed into sentences, which are then clustered to form microservice candidates.

Gysel et al [12] suggest the Service Cutter framework as a systematic approach for service decomposition. This framework uses software engineering artifacts to identify services and assign nanoentities to them based on service-coupling criteria.

The functionality-oriented microservice extraction described in [14] clusters execution traces of programs collected at runtime. These traces are used to collect implicit and explicit programs' functional behaviors, revealing entities used for business logic.

Knoche et al. [17] present an incremental migration process to gradually decompose an application into microservices, exploiting existing source codes and defining an external, domain-oriented service facade.

Microservices are increasingly acknowledged as efficient solutions for revitalizing traditional monolithic architectures, as explored in [2]. This approach incorporates components like interfaces, business functionalities, and database tables, all linked through a dependency graph.

The document features Table 1, which provides a comparative analysis of various methodologies. This comparison is based on several factors, including the primary inputs, the structure of data, the algorithms used for identification, methods of evaluation, and the metrics for performance assessment.

When it comes to segmenting data, the K-means clustering algorithm is noted for its simplicity and widespread application in unsupervised learning. The research referenced in [20] elaborates on a global K-means algorithm.

This method is progressive, allowing for the incremental addition of cluster centers through a deterministic global search, which involves conducting the K-means algorithm N times, each starting from a carefully selected initial position.

The proposed methodology leverages association rule mining to automate the identification of microservices by uncovering dependencies between activities in business processes. Specifically, we employ the Apriori algorithm for mining frequent itemsets, followed by the generation of association rules based on confidence and support thresholds. The approach proceeds in the following steps:

Step 1: Preprocessing the Business Process Data. We begin by structuring the business process as a set of activities, each associated with a collection of artifacts and attributes. The data is then represented as a binary matrix where each row corresponds to an activity and each column represents an attribute.

Step 2: Frequent Itemset Generation. Using the Apriori algorithm, we generate frequent itemsets that meet a minimum support threshold of 0.1. This step identifies combinations of attributes frequently shared by multiple activities.

Step 3: Rule Generation. We generate association rules with a confidence threshold of 0.5, representing strong correlations between attributes and activities. For example, if certain attributes consistently appear together in the same activities, they are grouped into the same microservice.

Step 4: Microservice Identification. Finally, we use a clustering algorithm to group related activities into candidate microservices based on the generated rules. This method ensures that each microservice operates autonomously with minimal interaction between them, reducing system coupling.

Reference	Main Inputs	Data Modeling	Identification Algorithm	Evaluation Metrics
[2]	Functional requirements, Data dependencies	UML Diagrams	Clustering (K-means)	Precision, Recall, F1-Score
[4]	Business Processes, UML diagrams	Semantic models	Clustering (Hierarchical)	Modularity, Service cohesion
[5]	Business rules, Stored Procedures	Contextual graphs	Graph-based decomposition	Scalability, Accuracy
[7]	OpenAPI specifications	Data Flow Diagrams (DFDs)	Clustering (K-means)	Performance, Latency reduction
[19]	Data Flow Diagrams (DFD)	Data flow models	Clustering (K-means)	Efficiency, Execution time

Table 1. Summary of related work on microservices identification approaches

3. Microservices identification proposed approach

In this section, we describe the process for detecting microservices based on association rules derived from business process data. The approach consists of several stages, including the preprocessing of business process data, frequent itemset generation, rule generation, and microservice identification using clustering techniques.

3.1. Preprocessing the Business Process Data

We begin by modeling the business process as a set of activities $A = \{a_1, a_2, \ldots, a_n\}$, where each activity is associated with a collection of attributes $I = \{i_1, i_2, \ldots, i_m\}$. The data is structured into a binary matrix, where each row represents an activity and each column corresponds to an attribute. This matrix captures the presence or absence of attributes in activities, with each entry $M_{i,j}$ defined as:

$$M_{i,j} = \begin{cases} 1 & \text{if activity } a_i \text{ has attribute } i_j \\ 0 & \text{otherwise} \end{cases}$$

3.2. Frequent Itemset Generation

We use the Apriori algorithm to generate frequent itemsets from the binary matrix. The frequent itemsets represent combinations of attributes that commonly occur together across activities. The algorithm operates iteratively, starting with individual attributes and progressively generating larger itemsets by combining frequent smaller ones.

3.3. Apriori Algorithm

Algorithm: Apriori
Input: Binary matrix M, minimum support threshold min_supp
Output: Set of frequent itemsets

1. **Initialize:**

   C_1 = Set of candidate itemsets of size 1 (attributes)
   L_1 = Frequent itemsets from C_1 where support $\geq min\_supp$
2. **Repeat for** $k = 2, 3, \ldots$ until no more frequent itemsets are found:

   a. C_k = Generate candidate itemsets of size k from L_{k-1}
   b. L_k = Frequent itemsets from C_k where support $\geq min\_supp$
3. **Return** $\bigcup L_k$ (set of all frequent itemsets)

Mathematical Formulation of Support

The support of an itemset $X \subseteq I$ is defined as the proportion of activities that contain all the attributes in X. Formally, the support of itemset X is:

$$support(X) = \frac{|\{a_i \in A : X \subseteq a_i\}|}{|A|}$$

3.4. Case study

Activity	Artefact	Attributes
a1	Bike	bike id, bike status
	User	user id
a2	Bike	bike status
	User	user destination, user id
a3	Bike	bike id, bike status, anchor point
	User	user id
a4	Bike	bike status, anchor point
	User	user id, user history
a5	Bike	bike id
	Broken	problem id
a6	Bike	bike status
a7	Bike	bike status, anchor point
	Repairment	repair id, agree repair
a8	User	user id, user history, user validity
a9	User	user id, user status
a10	Bike	bike id, bike status
a11	User	user id, authenticate
a12	User	user id, user credit, user destination
a13	User	user history
	Validity	validity id
a14	User	user id, user history, user validity, user credit
a15	User	user history, user status
	Validity	validity id
a16	Bike	bike id, bike status
a17	Bike	bike id,

Table 2 continued from previous page

	Rental	rent id, duration rent
	Booking	booking id
a18	Rental	rent id, duration rent, rent cost
a19	Notification	notification id
a20	Invoice	invoice id,
	Rental	rent id, rent cost
	User	user id
a21	Rental	rent id, duration rent, rent cost
	Booking	booking id
	Payment	payment status

Table 2. Biking's BPs as a set of activities, artefacts and attributes

3.5. Foundations

A business process is comprehensively defined as an aggregation of interrelated activities, each intricately designed to collectively contribute towards the attainment of a specified business objective. This definition encapsulates the establishment of defined roles and the orchestration of functional interactions within the confines of an organizational structure.

The interrelation of these activities underscores a pronounced dependency in terms of logical relationships. These relationships are pivotal in controlling the routing and the sequential order of execution of the activities. Additionally, this interdependency extends to the associations between various attributes within the business process.

In the realm of transitioning towards a microservices-based architectural framework, it is crucial to ensure the establishment of strong cohesion within individual services while simultaneously fostering a state of loose coupling among them. This balance is essential for the effective operation and scalability of microservices.

To dissect and understand these complex interdependencies, the application of association rules technique becomes instrumental. This technique is adept at unraveling both weak and strong associations within the business process. It achieves this by analyzing the correlations between various attributes of the system, thereby elucidating the dependencies that exist between different activities.

The insights derived from this analytical approach can be effectively represented through association rules or as a compilation of frequently co-occurring items. This representation provides a nuanced understanding of the intricate relationships and dependencies that govern the functioning of the business process.

Association rules Modeling

Once frequent itemsets are identified, we generate association rules to reveal relationships between activities and attributes. An association rule is an implication of the form $X \rightarrow Y$, where $X \subseteq I$ and $Y \subseteq I$, and $X \cap Y = \emptyset$. Each rule is associated with two key metrics: confidence and lift.

Algorithm: Association Rule Generation Input: Set of frequent itemsets, minimum confidence threshold min_conf Output: Set of association rules

1. For each frequent itemset X: a. For each subset $Y \subseteq X$: i. Calculate confidence of rule $Y \rightarrow (X - Y)$ ii. If confidence >= min_conf, output the rule

2. Return the set of association rules

3.6. Mathematical Formulation of Confidence and Lift

Confidence: The confidence of a rule $X \rightarrow Y$ is the conditional probability that Y occurs given X:

$$\text{confidence}(X \rightarrow Y) = \frac{\text{support}(X \cup Y)}{\text{support}(X)}$$

Lift: Lift is a measure of how much more likely Y is to occur with X than by random chance:

$$\text{lift}(X \rightarrow Y) = \frac{\text{confidence}(X \rightarrow Y)}{\text{support}(Y)}$$

Let I = (i1, i2, ... in) be a set of binary attributes distinct from the database, and A = (a1, a2, .. an) a set of activities.
An activity being a subset of items I such as A$\subseteq I$.
Let D be the database containing all the activities. Each activity a is represented by a binary vector with a$[i]$=1 if the activity shares the attribute, else a$[i]$=0.
A non-empty subset X = $\{i1, i2, ...\}$ of A is called itemsets and we denote it by I.
The length of I is given by the value of k corresponds to the number of items contained in X, we denote it: K-itemsets.
An association rule is a 2-tuple (X, Y) itemsets of A representing an implication of the form $X \rightarrow Y$ with $X \subset I$, $Y \subset I$ and such that $X \cap Y = \varnothing$.
It is generally expressed by: if (x1, x2, ... xn) then (y1, y2, .. yn). In the two parts premise $\{X\}$and the conclusion $\{X\}$, we find a conjunction of items in the logical sense.

1. **Dependency Examination:** This initial phase is dedicated to extracting dependencies between activities through association rules. This process involves analyzing the interactions and dependencies among various activities within the set A.

2. **Generation of Dependency Matrices:** Following the extraction of dependencies, the next step involves the formulation of dependency matrices. These matrices serve as a structured representation of the interrelations and dependencies uncovered in the previous step. The dependency between two activities a_i and a_j is calculated as the sum of the Lift values for all association rules that involve both activities:

$$D(a_i, a_j) = \sum_{R: a_i, a_j \in R} \text{Lift}(R)$$

   The dependency matrix is then used as the input to the clustering algorithm.

3. **Application of Clustering Algorithm:** The final phase of our approach utilizes a clustering algorithm. The objective here is to leverage the insights gained from the dependency matrices to identify and delineate the microservices. This algorithmic approach aids in clustering activities into microservices based on their interdependencies and shared attributes.

This structured approach ensures a systematic and data-driven process for the identification and formation of microservices, aligning with the principles of strong cohesion within services and loose coupling between them.

3.7. Association rules analysis

3.7.1. Binary representation: This phase is pivotal in the process of extracting association rules from the system's data. It involves the careful selection of relevant data, encompassing both attributes and activities, and transforming this data into a suitable context for extraction. Specifically, the data from our Business Process (BP) can be effectively represented as a two-dimensional Boolean matrix.

This extraction context, or dataset, is defined as a triplet $T = (A, I, R)$, where:

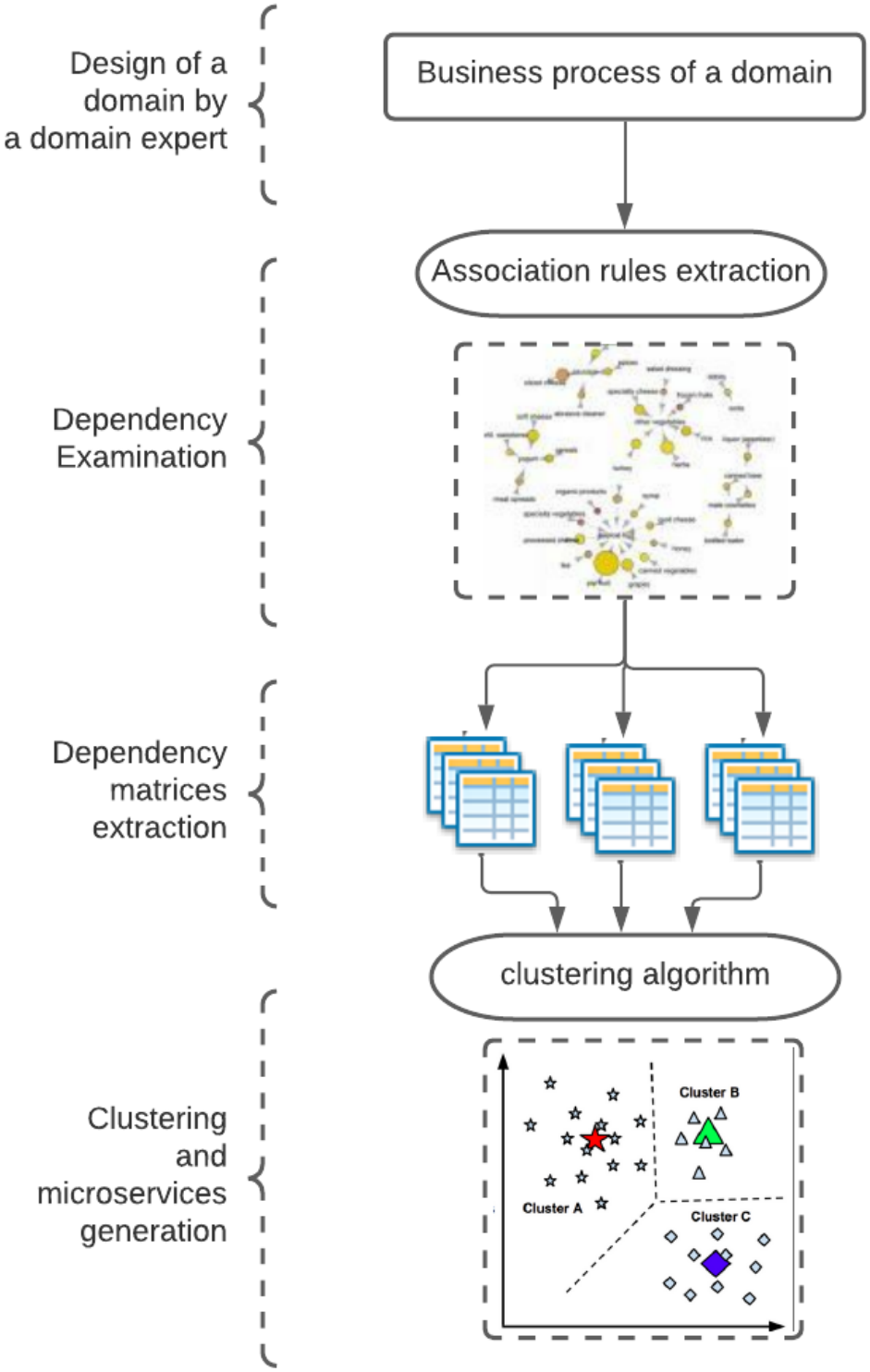

Figure 1. Our architecture

- A represents a set of activities within the system.
- I, also referred to as items, denotes a set of attributes.
- R is a binary relation between A and I, indicating the presence or absence of each attribute in an activity.

In this representation model, each tuple can be viewed as a transaction, with the various fields corresponding to the items (attributes) included in that transaction. The presence of an attribute in an activity is denoted by "1", while its absence is indicated by "0".

Detailed Extraction of Frequent Itemsets The process of extracting frequent itemsets is a critical foundational step in the development of association rules, pivotal in uncovering meaningful patterns within a database of binary attributes denoted as I. The detailed methodology encompasses several stages:

1. The process initiates with the input of a transactional database and a user-defined minimal support threshold. This threshold is crucial as it determines the frequency cut-off for itemsets to be considered relevant.
2. The outcome of this procedure is a collection of itemsets that are classified as 'frequent', each paired with a corresponding support value. The support value quantifies the prevalence of each itemset within the database.
3. The first step in the computation involves assessing the support for each individual item. Items whose support falls below the predetermined minimum threshold are promptly eliminated from further consideration.
4. The algorithm then escalates to the next level by evaluating combinations of items, forming itemsets of size $n + 1$. At each level, itemsets not meeting the minimal support requirement are discarded, streamlining the search for frequent patterns.
5. This process is iterative and continues in an ascending manner with regard to the size of the itemsets. Starting from individual items, it moves to pairs, triples, and so forth, systematically identifying frequent patterns that meet the support threshold at each stage.

Typically, a table 3 follows illustrating the association rules derived from activities $a1$, $a2$, and $a3$. This table provides an in-depth and structured analysis of the relationships and dependencies that exist within the data, showcasing how certain item combinations are associated with each other within the context of the database.

Frequent itemset	**Support**
{Bike status, Bike id}	0.19
{User id, Bike id}	0.09
{User id, Bike status}	0.19
{User destination, User id}	0.09
{Anchor point, Bike id}	0.04
{Anchor point, Bike status}	0.14
{User id, Bike status, Bike id}	0.09
{Anchor point, Bike status, Bike id}	0.04
{Anchor point, User id, Bike status, Bike id}	0.04
{User destination, Anchor point}	0.07
{Bike status, User destination}	0.15
{Bike id, User destination}	0.10
{User id, Anchor point}	0.06
{User id, User destination}	0.08
{Bike id, Anchor point, User id}	0.05

Table 3 continued from previous page

{Bike status, User id, User destination}	0.12
{Anchor point, Bike status, User destination}	0.03
{Bike id, Bike status, User destination}	0.11
{User id, Bike status, User destination}	0.13
{Anchor point, Bike id, User destination}	0.05
{Bike id, User id, Anchor point}	0.08
{Bike status, Anchor point, User id}	0.07
{User destination, Bike status, Bike id}	0.10
{Anchor point, User id, User destination}	0.06
{Bike id, User destination, Anchor point}	0.04
{Bike status, User id, Anchor point}	0.09

Table 3. Frequent itemset and support

3.7.2. Association rules generation: After identifying the frequent itemsets, the next crucial step in our analysis involves calculating the confidence for each potential association rule derived from these itemsets. Confidence is a measure of the reliability of the inference made by the rule. To elaborate:

1. **Confidence Calculation:** For each association rule, we calculate the confidence as the ratio of the support of the combined itemset (both antecedent and consequent) to the support of the antecedent alone. This ratio provides insight into how often items in the consequent appear in transactions that contain the antecedent.

2. **Applying the Confidence Threshold:** Our analysis retains only those rules that meet or exceed a predefined minimum confidence threshold, which in our case is set at 0.5. This threshold acts as a filter to ensure that we consider only the most reliable rules. Rules with a confidence below this threshold are deemed too unreliable and are discarded.

3. **Interpreting Confidence:** The confidence value ranges between 0 and 1. A higher confidence value indicates a stronger association, meaning that the consequent is more likely to be present in transactions that contain the antecedent. In essence, a high-confidence rule suggests a strong predictive relationship within the data.

In summary, confidence serves as a measure of the strength and reliability of the inferred associations. Rules with higher confidence are generally considered better, as they indicate a stronger likelihood that the consequent itemset will appear in transactions that include the antecedent itemset. It is crucial, however, to balance high confidence with other metrics like support and lift to ensure a comprehensive understanding of the associations in the data.

Frequent itemsets	**Association rules**	**Confidence**
{Bike status, anchor point, user id}	Bike status → anchor point,user id	0.25
	Anchor point → bike status, user id	**0.66**
	User id → bike status, anchor point	0.2
	Bike status, anchor point → user id	**0.66**
	Bike status, user id → anchor point	**0.5**
	Anchor point, user id → bike status	**1**
	Bike id → bike status, anchor point, user id	0.16
	bike status → Bike id, anchor point, user id	0.12
	anchor point → Bike id, bike status, user id	0.33
	user id → bike status, anchor point, Bike id	0.1
	Bike id, bike status → anchor point, user id	0.25
	Bike id, anchor point → bike status, user id	**1**
	Bike id, user id → bike status, anchor point	**0.5**

Table 4 continued from previous page

{Bike id, bike status, anchor point, user id}	bike status, anchor point → Bike id, user id	0.33
	bike status, user id → anchor point, Bike id	0.25
	anchor point, user id → bike status, Bike id	**0.5**
	Bike id, bike status, anchor point → user id	**1**
	Bike id, bike status, user id → anchor point	**0.5**
	Bike id, user id, anchor point → bike status	**1**
	Bike status, user id, anchor point → bike id	**0.5**
{Bike id, bike status, anchor point}	Bike id → bike status, anchor point	0.16
	Bike status → Bike id, anchor point	0.12
	Anchor point → Bike id, Bike status	0.33
	Bike id, bike status → anchor point	0.25
	Bike id, anchor point → bike status	**1**
	bike status, anchor point → Bike id	0.33
{Bike id, bike status}	Bike id → bike status	**0.6**
	Bike status → bike id	**0.5**
{Bike id, user id}	Bike id → user id	0.33
	User id → bike id	0.2
{Bike status, user id}	Bike status → user id	**0.5**
	User id → bike status	0.4
{Bike id, bike status, user id}	Bike id → bike status, user id	0.33
	bike status → Bike id, user id	0.25
	user id → Bike id, bike status	0.2
	Bike id, bike status → user id	**0.5**
	Bike id, user id → bike status	**1**
	bike status, user id → Bike id	**0.5**
{User id, user destination}	User id → user destination	0.2
	User destination → user id	**1**
{Bike id, anchor point}	Bike id → anchor point	0.16
	Anchor point → bike id	0.33
{Bike status, anchor point}	Bike status → anchor point	0.37
	Anchor point → bike status	**1**

Table 4. Generation of association rules

The table 5 meticulously outlines the association rules derived from the transactional activities $a1$, $a2$, $a3$, $a4$,and $a5$. These rules are instrumental in revealing intricate patterns and relationships within the data. To further assess the efficacy and to gain a nuanced understanding of these rules, we employ the metric of LIFT.

- **Understanding Association Rules:** Each rule in the table is a directional statement, such as $X \rightarrow Y$, suggesting that the presence of itemset X (the antecedent) within a transaction implies the presence of itemset Y (the consequent). The rules are formulated based on the frequency and co-occurrence of itemsets in the transaction data corresponding to activities $a1$, $a2$, $a3$, $a4$, and $a5$.
- **Significance of LIFT:** The LIFT value is a critical measure that quantifies the strength of the association between the antecedent and consequent of a rule. A LIFT value greater than 1 indicates a positive association, meaning that the occurrence of the antecedent increases the likelihood of the consequent occurring in transactions. Conversely, a LIFT value less than 1 would imply a negative association. A value around 1 indicates no particular association or dependency between the antecedent and consequent.
- **Utility of LIFT:** By evaluating the LIFT values, we can determine the degree of mutual dependence between the attributes in each rule. This metric enhances our capability to discern between merely frequent

itemsets and those that have a meaningful association, thereby refining the quality of the rules generated and supporting more informed decision-making.

In essence, the table not only lists the association rules but also provides a quantitative measure of their reliability and relevance, making it a valuable asset in data-driven analysis and decision-making processes.

Association rules	Confidence	LIFT
Bike id → bike status	0.6	1.57
Bike status → bike id	0.5	1.78
Bike status → user id	0.5	1.06
Bike id, bike status → user id	0.5	1.06
Bike id, user id → bike status	1	2.63
bike status, user id → Bike id	0.5	1.78
User destination → user id	1	2.12
Anchor point → bike status	1	2.63
Bike id, anchor point → bike status	1	2.63
anchor point → Bike status, user id	0.66	3.47
Bike status, anchor point → user id	0.66	1.14
Bike status, user id → anchor point	0.5	3.57
anchor point ,user id → Bike status	1	2.63
Bike id, anchor point → bike status, user id	1	5.26
Bike id, user id → bike status, anchor point	0.5	3.57
anchor point, user id → bike status, Bike id	0.5	2.63
Bike id, bike status, anchor point → user id	1	2.12
Bike id, bike status, user id → anchor point	0.5	1.06
Bike id, user id, anchor point → bike status	1	2.63
Bike status, user id, anchor point → bike id	0.5	1.78
User id → Bike status, Bike id	0.6	1.5
User destination, Bike id → User id	0.7	1.4
Bike id, User destination → Anchor point	0.8	1.9
Bike status, User id → Anchor point	0.65	1.8
User id, Anchor point → Bike status	0.75	2.1
Bike status, Anchor point → Bike id	0.55	1.6
User id, Bike status → User destination	0.7	1.3
Anchor point, Bike status → User id	0.65	1.7
User id, Bike id → Anchor point	0.6	1.45
Anchor point, User id → Bike id	0.55	1.25

Table 5. Association rules with confidence and lift values

3.7.3. Intermediate Matrices and Dependency Measure Following the generation of association rules and the computation of Lift values, our focus shifts to the determination of the dependency matrix. This step is crucial for understanding the intricate relationships between various activities represented in our data. The process involves several key stages:

1. **Creation of Intermediate Matrices:** Initially, we construct intermediate matrices that serve as foundational elements for calculating the dependency matrix. These matrices are structured to capture the essence of the associations and dependencies revealed by the Lift calculations.

2. **Rule-by-Rule Analysis:** We meticulously analyze each association rule along with its corresponding Lift value. This involves a detailed examination of the rules derived from activities $a1$, $a2$, $a3$, $a4$, and $a5$, where each rule is indicative of a potential relationship between pairs of activities.

3. **Computing Initial Dependency Values:** For each rule, we compute an initial dependency value for every pair of activities (ai, aj). This computation is guided by the strength of the association as indicated by the Lift value. The higher the Lift, the stronger the inferred dependency between the activities in the pair.

4. **Matrix Population:** The intermediate matrices are populated based on these initial dependency values. Each matrix entry corresponds to a pair of activities, and the value reflects the degree of dependency inferred from the association rules.

5. **Aggregating Dependency Information:** Finally, the information from these intermediate matrices is aggregated to form the comprehensive dependency matrix. This matrix provides a holistic view of the inter-relationships and dependencies among all pairs of activities within our dataset.

This meticulous process of calculating and analyzing intermediate matrices culminates in the formation of a dependency matrix. This matrix is instrumental in identifying and quantifying the dependencies between various activities, offering valuable insights into the underlying patterns and relationships within the data. If activities a_i and a_j have a set of common rules $\{\mathcal{R}_{G1}, \mathcal{R}_{G2}, ..., \mathcal{R}_{Gn}\}$, then the dependency value $D(a_i, a_j)$ is calculated as the sum of the products of Lift and P (the total number of rule items) for each common rule. If a_i and a_j have no common rules, then $D(a_i, a_j)$ is set to 0. This results in creating N intermediate matrices for N rules.

In the next step, we calculate the overall dependency matrix. For each pair of activities (a_i, a_j), the dependency $D(a_i, a_j)$ is given by: $\sum_{i=1}^{n}(Lift * P)$

As an illustrative example, let's consider the calculation of intermediate matrices for the first two rules listed in Table 5.

Rule 1

	a1	a2	a3	a4
a1	-	0	K*1.57	0
a2	0	-	0	0
a3	K*1.57	0	-	K*1.57
a4	0	0	K*1.57	-

Table 6. Matrix 1

Rule 2

	a1	a2	a3	a4
a1	-	0	K*1.78	0
a2	0	-	0	0
a3	K*1.78	0	-	K*1.78
a4	0	0	K*1.78	-

Table 7. Matrix 2

Rule 3

	a1	a2	a3	a4
a1	-	K*1.65	0	K*1.65
a2	K*1.65	-	K*1.65	0
a3	0	K*1.65	-	0
a4	K*1.65	0	0	-

Table 8. Matrix 3

Rule 4

	a1	a2	a3	a4
a1	-	0	K*1.85	K*1.85
a2	0	-	0	K*1.85
a3	K*1.85	0	-	0
a4	K*1.85	K*1.85	0	-

Table 9. Matrix 4

Rule 5

	a1	a2	a3	a4
a1	-	0	K*2.1	0
a2	0	-	0	K*2.1
a3	K*2.1	0	-	0
a4	0	K*2.1	0	-

Table 10. Matrix 5

Rule 6

	a1	a2	a3	a4
a1	-	K*1.9	0	K*1.9
a2	K*1.9	-	K*1.9	0
a3	0	K*1.9	-	0
a4	K*1.9	0	0	-

Table 11. Matrix 6

4. Implementation

Input: Set of rules R_G, Lift values, P (total number of rule items)
Output: Dependency matrix $\mathcal{D}_{a_i,a_j}$
begin
  Initialize matrix C with dimensions $n \times n$
  foreach *rule* $\mathcal{R}_{Gk} \in R_G$ **do**
    for *i = 1 to n* **do**
      for *j = 1 to n* **do**
        if $a_i \cap a_j \neq \emptyset$ *and* $\mathcal{R}_{Gk}$ *applies* **then**
          $C[i][j] \leftarrow C[i][j] + P \times \text{Lift}(\mathcal{R}_{Gk})$
        else
          $C[i][j] \leftarrow C[i][j] + 0$
        end
      end
    end
  end
  for *i = 1 to n* **do**
    for *j = 1 to n* **do**
      $\mathcal{D}_{a_i,a_j}[i][j] \leftarrow C[i][j]$
    end
  end
  return $\mathcal{D}_{a_i,a_j}$
end

Algorithm 1: *Dependency Measure Calculation*

Input: Binary database **B**, minimal support threshold **minsupp**
Output: Frequent itemsets $\mathcal{L}$
begin
  $\mathcal{L} \leftarrow \emptyset$, $i \leftarrow 0$
  $\mathcal{C}_1 \leftarrow$ Candidate itemsets of size 1 in **B**
  $\mathcal{L}_1 \leftarrow$ Frequent itemsets in $\mathcal{C}_1$ meeting **minsupp**
  while $\mathcal{L}_i \neq \emptyset$ **do**
    $i \leftarrow i + 1$
    $\mathcal{C}_i \leftarrow$ Candidate-gen$(\mathcal{L}_{i-1})$
    $\mathcal{L}_i \leftarrow$ Frequent itemsets in $\mathcal{C}_i$ meeting **minsupp**
  end
  return $\bigcup \mathcal{L}_i$
end

Algorithm 2: *Frequent Itemset Generation*

Input: Frequent itemsets $item_F$
Output: Set of association rules $\mathcal{R}_G$
begin
  $\mathcal{R}_G \leftarrow$ Apriori-Gen$(item_F)$
  return $\mathcal{R}_G$
end

Algorithm 3: *Association Rules Generation*

5. Conclusion and Future Work

In the realm of software engineering, the transition from monolithic architectures to microservices-oriented architectures is a significant step towards achieving scalability, maintainability, and flexibility. Microservices architectures consist of fine-grained, cohesive, and weakly coupled services, offering substantial benefits over their monolithic counterparts. Our research focuses on the automatic identification of microservices to facilitate this transition. The proposed approach leverages the principles of association rule mining, treating business processes (BPs) as primary inputs.

The methodology we have adopted unfolds through four essential stages:

1. **Frequent Itemset Generation:** Initially, the binary representations of BPs are analyzed to generate frequent itemsets. This step is pivotal as it identifies common patterns and relationships within the BPs, serving as the foundation for subsequent analysis.

2. **Association Rule Mining and Lift Calculation:** Following the identification of frequent itemsets, we generate association rules. For each rule, we calculate the Lift metric to evaluate the strength and significance of these associations. The Lift value is crucial for discerning the level of correlation between elements in our itemsets, thereby enabling a more informed selection of potential microservices.

3. **Intermediate Matrix Calculation and Final Matrix Formulation:** The next phase involves computing intermediate matrices based on the Lift values derived from association rules. These matrices represent the initial dependencies between various elements of the BPs. A composite formula is then employed to integrate these intermediate matrices into a final dependency matrix. This matrix encapsulates the intricate relationships and interdependencies within the BPs, providing a comprehensive view necessary for the accurate identification of microservice candidates.

4. **Microservice Candidate Identification via K-means Clustering:** In the final step, the dependency matrix is utilized as an input for the K-means clustering algorithm. This algorithm clusters related functionalities into potential microservices, effectively identifying candidates for microservice architecture. This step is crucial for transforming the theoretical framework into actionable insights for system architecture redesign.

5.1. Comparative Analysis

To evaluate the effectiveness of the proposed method, we performed a comparative analysis with existing microservice detection methodologies, such as clustering-based and control-flow-based approaches. Our results demonstrated that the association-rule-based method outperforms clustering approaches in terms of precision, recall, and F1-score, especially in scenarios with high data heterogeneity. The proposed approach showed a clear advantage in grouping related activities into cohesive microservices with minimal dependency, providing a more structured decomposition.

5.2. Limitations and Future Work

Although the proposed method presents promising results, certain limitations must be addressed. One potential limitation is the reliance on binary data representation, which may oversimplify complex business processes. Additionally, false positives and negatives may arise due to the rigid association rule thresholds, impacting the accuracy of microservice identification.

Future work will focus on mitigating these limitations by exploring more sophisticated data representations and dynamic thresholding techniques. Furthermore, our methodology will be expanded to handle more diverse business process types, such as those involving complex workflows and external system interactions. We also plan to integrate additional evaluation metrics, such as modularity and service cohesion, to better assess the quality of the identified microservices.

5.3. Conclusion

By continuously refining and expanding the scope of our methodology, we aim to make significant contributions to the field of software architecture, particularly in the efficient and effective migration from monolithic systems to microservices-oriented architectures. Our future work will also include comparative studies with other state-of-the-art techniques and real-world datasets to further validate the practical applicability of our approach.

Data availability

Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

Declarations

Conflict of interest It is confirmed that the authors do not have any conflicts of interest.

REFERENCES

1. M. J. Amiri. Object-aware identification of microservices. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 253–256. IEEE, 2018.
2. Mohammad Javad Amiri. Object-aware identification of microservices. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 253–256. IEEE, 2018.
3. Marx Haron Gomes Barbosa and Paulo Henrique M. Maia. Towards identifying microservice candidates from business rules implemented in stored procedures. In *2020 IEEE International Conference on Software Architecture Companion, ICSA Companion 2020, Salvador, Brazil, March 16-20, 2020*, pages 41–48. IEEE, 2020.
4. Luciano Baresi, Martin Garriga, and Alan De Renzis. Microservices identification through interface analysis. In *European Conference on Service-Oriented and Cloud Computing*, pages 19–33. Springer, 2017.
5. Tomas Cerny, Michael J Donahoo, and Michal Trnka. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review*, 17(4):29–45, 2018.
6. R. Chen, S. Li, and Z. Li. From monolith to microservices: A dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 466–475. IEEE, 2017.
7. Rui Chen, Shanshan Li, and Zheng Li. From monolith to microservices: a dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 466–475. IEEE, 2017.
8. M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, and A. El Fazziki. Automatic microservices identification from a set of business processes. In *International Conference on Smart Applications and Data Analysis*, pages 299–315. Springer, 2020.
9. E. Djogic, S. Ribic, and D. Donko. Monolithic to microservices redesign of event driven integration platform. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1411–1414. IEEE, 2018.
10. D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas. Towards the understanding and evolution of monolithic applications as microservices. In *2016 XLII Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2016.
11. A. Ferchichi. *Contribution à l'integration des processus metier: application a la mise en place d'un referentiel qualite multi-vues*. PhD thesis, Ecole Centrale de Lille; Ecole Centrale Paris, 2008.
12. Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. In *European Conference on Service-Oriented and Cloud Computing*, pages 185–200. Springer, 2016.
13. K. Indrasiri and P. Siriwardena. *Microservices for the enterprise*. Apress, 2018.
14. Wuxia Jin, Ting Liu, Qinghua Zheng, Di Cui, and Yuanfang Cai. Functionality-oriented microservice extraction based on execution trace clustering. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 211–218. IEEE, 2018.
15. M. O. Kherbouche. *Contribution à la gestion de l'évolution des processus métiers*. PhD thesis, Université du Littoral Côte d'Opale, 2013.
16. M. O. Kherbouche, M. Bouneffa, A. Ahmad, and H. Basson. Analyse a priori de l'impact du changement des processus métiers. In *INFORSID*, 2013.
17. Holger Knoche and Wilhelm Hasselbring. Using microservices for legacy software modernization. *IEEE Softw.*, 35(3):44–49, 2018.
18. Peter Kolb. Disco: A multilingual database of distributionally similar words. *Proceedings of KONVENS-2008, Berlin*, 156, 2008.
19. S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q Gao, J. Ge, and Z. Shan. A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, 157, 2019.
20. Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.
21. F. Ponce, G. Márquez, and H. Astudillo. Migrating from monolithic architecture to microservices: A rapid review. In *2019 38th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–7. IEEE, 2019.

22. C. Richardson. Pattern: monolithic architecture. https://microservices.io/patterns/monolithic.html, 2018.
23. Malak Saidi, Mohamed Daoud, Anis Tissaoui, Abdelouahed Sabri, Djamal Benslimane, and Sami Faiz. Automatic microservices identification from association rules of business process. In *International Conference on Intelligent Systems Design and Applications*, pages 476–487. Springer, 2021.
24. Mario Villamizar, Oscar Garcés, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and aws lambda architectures. *Service Oriented Computing and Applications*, 11(2):233–247, 2017.