Towards Migrating Legacy Software Systems to Microservice-based Architectures: a Data-Centric Process for Microservice Identification

Yamina Romani, Okba Tibermacine, Chouki Tibermacine

Towards Migrating Legacy Software Systems to Microservice-based Architectures: a Data-Centric Process for Microservice Identification

Yamina Romani*, Okba Tibermacine*, Chouki Tibermacine§
* Computer science department, University of Biskra, Algeria
{yamina.romani, o.tibermacine}@univ-biskra.dz
§LIRMM, Univ Montpellier, CNRS, France
chouki.tibermacine@lirmm.fr

***Abstract*—"Microservice-based architecture" is an architectural style exploited to develop software systems with the main concern of independent maintainability, deployability and scalability. These important capabilities in modern software development and operation settings led many companies to migrate their existing (legacy) monolithic software systems towards microservice-based architectures. The migration process is a challenging task. It requires splitting the system into consistent parts that represent the set of microservices. Existing works focus mainly on functional aspects in this splitting. We argue in this work that it would be beneficial to start this splitting by decomposing the database into clusters, where the data in each cluster is associated to a microservice's own independent database. This is commonly known as the "database-per-service" pattern in microservice architectures. This paper proposes our preliminary work on a data-centric process to identify microservices. This process performs database schema analysis and clustering in order to make topic identification. It aims at identifying a set of topics which correspond to potential microservices.**

***Index Terms*—Microservices, Database-per-service pattern, Software Architecture, monolithic to microservice migration, topic identification, clustering.**

I. Introduction

The context of this work is software maintenance and reengineering and more particularly the migration of legacy systems into modern microservice architectures. This architecture style offers many qualities like maintainability, agility in development and delivery, and independent deployability & scalablity. Many companies develop their applications by instantiating this style. But a lot of legacy applications, still profit-making for their owners, need to be migrated instead of being redeveloped from scratch [9], [2]. This migration is conducted according to a process that can be divided roughly into two phases: microservice identification and code transformation/refactoring.

A plethora of works have been conducted these last years on this subject [10]. Most of these works tackle the problem of microservice identification from a business-logic point of view, using as a driver functional points (see related works for details). In this work, we propose to rearchitect a monolithic software system to a set of microservices from a **data-centric** point of view. We focus on splitting the data model (e.g. a database schema) to a set of cohesive and semantically related sub-models (i.e. a set of tables/documents) that are packaged with their corresponding business logic to a set of microservices. We argue that this way of splitting enables a better modularization of the software system. Indeed, in most development settings, the design of data is one of the first conducted activities. Considering this artifact (i.e. data models) as a starting point will make it easy to drive the rest of the software system splitting. In this way, we obtain a set of microservices each of which having its own data, being thereby able to host them on whatever storage service, to implement/migrate them with/to whatever DB-technology vendor, or to populate them from whatever sources.

We present in this paper the first step towards this goal. We propose a microservice identification process that analyses a database model, pre-processes its symbols and enriches these symbols with their semantically-related terms using lexical databases (e.g. WordNet / WordWeb), and finally classifies these symbols into a set of clusters that we consider as labels for microservice candidates. The process finishes by assigning database sub-models (sets of related tables/documents) to each identified microservice. In our work, the problem of microservice identification is seen as a problem of **topic identification/modeling** using an enriched set of symbols describing the data of the software system. The upcoming steps consist of migrating data and rewriting queries or persistence API client code, refactoring the code related to the data model (entities and repositories), the business logic (services and controllers) and the front-end of each microservice. These steps are not covered in this paper.

The remaining of the paper is organized as follows: Section II presents in depth the proposed data-centric microservice identification process. Section III provides the application of the process on a real-world small monolithic application. Section IV discusses related work and Section V concludes the paper and sketches the forthcoming work.

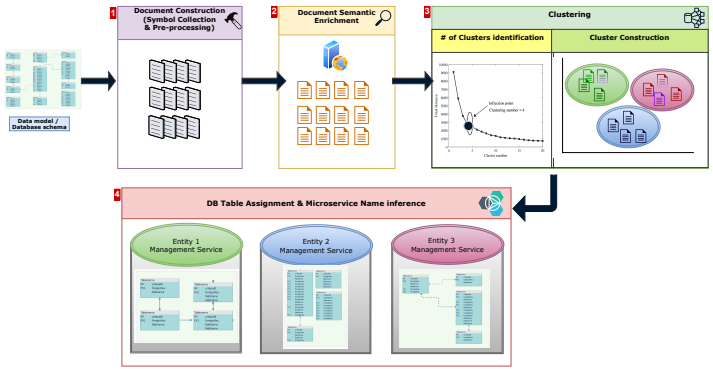

Fig. 1. Proposed Process for Microservice Indentification.

II. Proposed Process

In this section, we present the proposed process to identify microservices. Fig. 1 depicts this process which takes as input a data model of the monolithic software system. This can be an Entity-Relationship/class diagram or a SQL/No-SQL database schema. In order to process this data model, we propose to consider each table as a symbol document and the set of tables as a collection of documents.

The first step in this process is the document construction and pre-processing, where we pre-process the collected symbols from the database schema, which are the names of tables and their attributes/columns. At the end of this step, we obtain a set of cleaned documents.

The second step is Document Enrichment, which is intended to extend the words of each document with their synonyms and other semantically-related symbols.

The next step is clustering. It involves the document vectorization, the identification of the optimal number of clusters and at last classification. Finally, the output is a set of clusters that represent groups of symbols corresponding to topics that we consider as the set of labels for the potential microservices.

The last step refers to the microservice name inference based on the generated clusters of symbols. In the following subsections, we explain in depth all these steps.

A. Document Construction and Pre-processing

Based on the database schema model of the monolithic software system, we create a set of documents. Each document represents a collection of symbols derived from a given relational table (or No-SQL document[1]) in the data model. A symbol can be a table's name, a table's attribute, or a relationship between two tables. Each symbol is "modeled" according to its importance, where:

- The table's name is repeated many times in the document. This repetition gives it a heavier weight, since it is an important symbol characterizing a given data structure.
- A relationship between tables represents the existence of a foreign key (or a reference) which is a significant symbol that describes a functional or structural relationship between tables.
  According to our analysis of some database schema models, we distinguish two types of foreign keys.
  1) The first one is a singleton foreign key. It relates a source table to only a single target table. This type describes a strong relationship between the two documents. So, we model it by duplicating the foreign key and the name of the source document many times in the target document. Similarly, we duplicate the name of the target document in the source document. An example that illustrates the singleton foreign key is presented in Figure 2, where the "country" table links only the "customer" table with a foreign key.
  2) The second type is a non singleton foreign key, it relates the source table with more than a single target table.
     This type indicates a less strong relationship between the documents compared with the first type. Consequently, we model it by repeating the foreign key and the name of the source document less times than the first type in the target documents. We do the same for the name of the target documents in the source document. We have an example of this type of foreign keys in Figure 2: the "customer" table that is linked to two tables, the "shopping cart" and "orders" with a non singleton foreign key.

Note that the number of times these symbols are repeated is fixed empirically.

After this step, the document symbols are processed and cleaned using some NLP (*Natural Language Processing*) techniques as follows:

Tokenization, this procedure takes each document symbol and divides it into separate words (tokens). The separation is based on the existence of some characters between tokens like - and _ or capital letters [11].

Lemmatization, it is the process of transforming each word into its lemma (root) using a morphological analysis. For example, the words `studies` or `studying` are transformed into `study`, which is their root [11].

B. Document Semantic Enrichment

Since we use unsupervised machine learning that requires a lot of data to produce good results, we need to extend our documents (a technique known as "data augmentation"), by enriching them with semantically-related symbols. These symbols are related to our symbols by the following two links: "part of" and "type of", which are the mostly pertinent symbol relations for our problem of data modeling.

This enrichment step uses a lexical database, like WordNet.

[1]The reader should not confuse the "document" constructed in our process, which is composed of symbols (names/identifiers), and the document that is described in a No-SQL database, which is a data structure in this kind of databases. For simplicity reasons, we use the term *Table* to refer to both a relational database table and a No-SQL document.

C. Clustering

In this step, we classify all the symbols in clusters, where each cluster contains the most similar symbols.

In our process the clustering stage consists of three steps:

Document Vectorization, it is the procedure of transforming each document symbol into a vector space model. This procedure provides a weight for each symbol, which indicates the value of this symbol in its document and also considering the collection of all documents.

Number of clusters identification. Several clustering techniques require the number of desired clusters as an input. Thus, in this step, we calculate the optimal number of clusters according to our input data using an appropriate method such as Elbow [8] or Silhouette [5].

Cluster construction, it is the leading step that requires the selection of an appropriate clustering technique, such as hierarchical clustering or K-Means.

D. Database Table Assignment and Service Name inference

The result of the previous step is a set of clusters containing simple symbols. In this step, we distribute database tables into their corresponding clusters. The simplest case is where the name of a table and all its attributes/columns are in the same cluster. In this case, we assign that table to this cluster. But sometimes, clustering may rise some issues if for example, a given table name together with a subset of of its column names are in one cluster and the other column names of this table are in a different cluster. In this case, the developer should decide whether : i) we ignore this distribution of column names into different clusters and thereby group them in the cluster where the name of the table appears, or if ii) we split the table into distinct tables according to the result of clustering. At the end of this step, database tables are assigned into clusters.

Consequently, each potential microservice will be named with the dominant symbol in the related cluster. By convention, if the symbol name corresponds to a well-identified entity, like `Product` or `Customer`, the microservice will be named by appending the symbol with the word `ManagementService`, like `CustomerManagementService`.

III. Illustrative Example

To illustrate our proposed process for microservice identification, we used an E-commerce application called auto-parts [1]. The application was written with Java using the Spring boot framework. It consists of three modules (Admin, Client, Library), 63 classes, 14 interfaces and 14 database tables.

We first examined the database schema model of this application, then we collected all the extracted symbols in a text file based on the previously explained steps (in the Document Construction and Pre-processing steps). Figure 2 shows the Entity-Relationship Diagram of the application's database. After that, we cleaned the set of documents in the generated file using the NLP techniques as explained before.

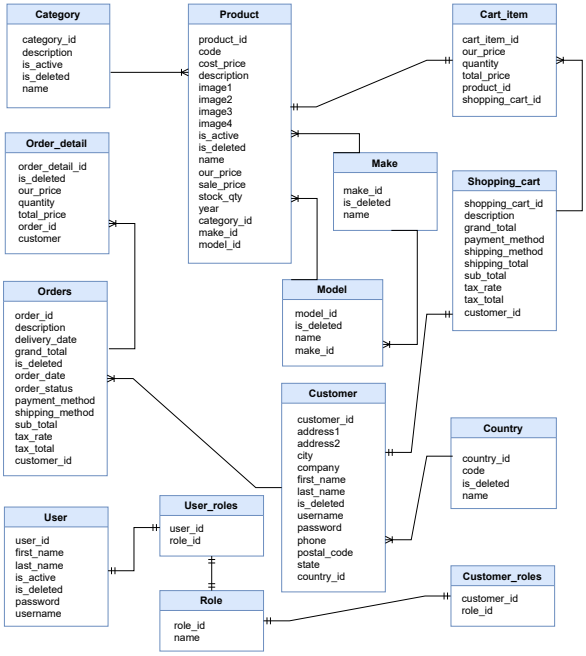

Fig. 2. Entity Relationship Diagram of the Auto-parts Application

Then, we enriched our document symbols focusing on the synonyms of the table's name which is considered as a significant term in the document. We used the WordWeb[2] English dictionary and thesaurus to obtain the semantic relationships of the table names "part of" and "type of".

The next step is clustering. We first transform the pre-processed documents into a set of vectors. We used TF-IDF method [5] to vectorize our documents. We chose this method because it gave us good results with many testing examples.

TF-IDF is measured using two scores:

TF(t) = (Number of times term/symbol t appears in a document) / (Total number of terms in the document).

IDF(t) = loge (Total number of documents / Number of documents with term t in it).

Then, TF-IDF is calculated as follows:

TF-IDF(t)=TF(t)*IDF(t). Finally, we obtain a set of vectors that represent the weights of the terms in each document.

Then, for the purpose of identifying the optimal number of clusters, we exploited a widely used method named "Elbow". This method calculates the cost function generated using several values of k (number of clusters) to the sum of the squared distance between the documents and their assigned cluster centroids (this corresponds to a measure of variance in the input data that we want to minimize through clustering).

Typically we choose the number of clusters where the curve of the sum of squared distance begins to form an elbow [8]. Figure 3 shows the curve obtained with Elbow method. Based on this result we select five as the number of clusters.

The last step of this stage relies on a clustering method. The k-means clustering algorithm was used to group the documents

[2]https://www.wordwebonline.com/

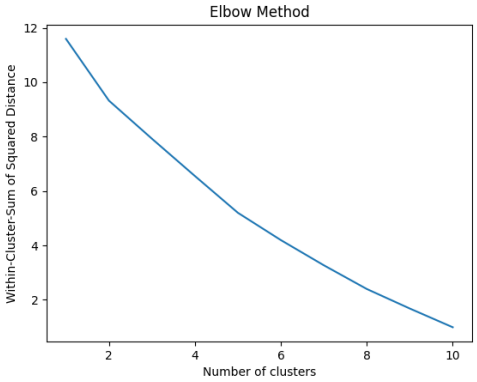

Fig. 3. Elbow Result

represented by the generated vectors. K-means is a straightforward method for grouping a collection of observations according to a specified number of clusters (k) [4].

The basic idea in this algorithm is to select randomly k centroids that correspond to the number of clusters, then it assigns each observation to the closest center based on similarity. Then the algorithm iterates by updating the centroids until no further changes to the partitioning are seen. Table 1 shows the result of clustering where each cluster consists of a set of documents (Tables), as well as a set of topics that symbolize each cluster from which the name of the microservice is selected.

According to the obtained results[3], we observed that each cluster consists of a set of tables that are working perfectly together. We are aware that this will not be always the case. In some situations, splitting large tables should be considered. This is why we proposed in our process that the developer intervenes to validate the clusters.

IV. Related Work

In the literature, different approaches for microservice identification and monolithic software decomposition have been recently conducted. Selmadji et al. [16] proposed a semi-automatic approach for microservice identification from monolithic Object-Oriented (OO) applications based on source code examination and the engineering's knowledge of the system to migrate. The goal of this migration is to make monolithic applications deployable on the cloud and respect DevOps practices. This approach splits the monolithic software classes based on the micro service characteristics using the ISO/IEC 25010:2011 model to measure the relevance of the obtained microservices.

Carvalho et al. [7] presented a multi-objective search-based approach to obtain microservices automatically from a legacy system. The approach takes into account five criteria: cohesion, coupling, feature modularization, reuse and communication overhead. The input of this approach is the source code of a Java legacy system, a list of features related to each execution of this legacy system and the number of microservices to be identified. The source code is represented as a graph where each vertex indicates a method of the legacy system corresponding to its respective feature and each edge indicates a relationship between methods. Then, Non-dominated Sorting Genetic Algorithm III (NSGA-III) was applied to identify the microsevices based on the objective function of each criterion.

Kalia et al. [12] presented the Mono2Micro approach developed at IBM to migrate monolithic applications towards a microservice architecture. The purpose of this migration is to shift enterprise production workloads to the cloud and benefit from its capabilities. Mono2Micro employs a hierarchical spatio-temporal decomposition using well-defined business use cases (the space dimension) and its runtime traces (time dimension) to partition the application classes. This approach takes into consideration cohesion and coupling criteria to split Java legacy software systems.

Barbosa et al. [3] introduced a manual approach to identify microservice candidates using business rules which are executed in stored procedures. This technique identifies the system requirements using an expert and examines the source code to map the stored procedures that correspond to the implementation of these business requirements. Then, database artifacts (stored procedures) are analysed to detect all business rules, where the rules that handle the same business requirements should be merged in the same microservices.

In Zhang et al. [17], the authors present an automated microservice identification approach that aims to partition software systems into parts using execution and performance logs of the legacy system. It considers two types of objects: controller objects (COs) and subordinate objects (SOs). Based on the measured relation between each pair of CO and SO, the system classes are grouped into microservices using NSGA by optimizing the objectives associated with functional (coupling and cohesion) and non functional (load balance) metrics.

Shanshan et al. [14] presented a semi-automatic decomposition of a given monolithic software system into microservices using dataflow diagrams. This approach creates a use case and business logic specification by means of a business requirement analysis, then, it generates the fine-grained Data Flow Diagrams (DFD) and the process-datastore version of this DFD which designates the business logic. The dependencies between processes and data stores are extracted into decomposable sentence sets that are grouped into individual modules to formulate the microservice candidates.

[16], [7], [12] and [17] take a functional perspective to split a monolithic software into microservices. Each study uses its own interpretation of microservices to evaluate the microservice relevance. Per contra, authors in [3] and [14] combine business-logic view and data flow exploitation to identify microservice candidates in monoliths.

Topic Modelling has been adopted in [6] to detect microservices based on domain terms, where it extracts relevant terms from the source code and uses them to obtain topics. Then a weighted graph is generated based on the structural dependencies and the distribution of topics among classes. At last, microservices are identified by applying the Louvain

[3]Source code, documents and results can be found here: https://bit.ly/31ySia7

Clusters	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
First three topics	country, customer, region	user, role, customer	order, detail, commercial	cart, shopping, item	category, make, product
Tables	customer country	user role customer role user role	order order detail	cart item shopping cart	product category make model
Microservice Name	CustomerManagement Service	UserManagement Service	OrderManagement Service	CartManagement Service	ProductManagement Service

TABLE I
CLUSTERING RESULTS

community detection algorithm on the generated graph.

In contrast to the previous works which are based on source code analysis or which take as drivers functional aspects, our proposal takes a data model-centric perspective. The identification of microservice candidates is performed using the data model, which is an artifact that exists in all situations in enterprise applications. We argue that relying solely on these artifacts as a starting point simplifies greatly microservice identification.

In [15], Newman proposed solutions (patterns) to decompose manually a database to fit a microservice architecture. This work considers a set of *already identified* microservices; it does not deal with their identification. In another work [13], the authors take databases into consideration, where they regard the monolithic enterprise system as a set of small subsystems, each of which has its own business responsibilities and its own data. This process is based on building a dependency graph for each subsystem, which is used to identify microservice candidates (dependencies include relations in source code and between database tables). The main limitation of this work is that it does not migrate all subsystems. Those that are independent (with no external dependencies) are transformed into microservices and the others remain as a monolith.

V. CONCLUSION

We presented a data-centric process for the identification of microservice candidates as a first step for migrating legacy software systems into a microservice-based architecture. The process relies on topic modelling applied to documents constructed from data model symbols and enriched by semantically related words. We applied the process on a small real-world example to show how it can be used for identifying microservice candidates. Though we used in the illustrative example WordWeb dictionary, Elbow and K-means methods in the process of microservice identification, the proposed process is generic and other dictionaries and clustering techniques can be used.

The ongoing work consists of proposing a process for automating as much as possible data migration by rewriting queries, and refactoring the code related to the data model, in addition to the rest of the application for each microservice.

REFERENCES

[1] Creative tech house website. http://www.creativetechhouse.com/project-details/4/e-commerce-auto-parts-java-thymeleaf-spring-boot-jpa-mysql-hybernate-application. Accessed: 2021-11-12.

[2] Florian Auer, Valentina Lenarduzzi, Michael Felderer, and Davide Taibi. From monolithic systems to microservices: An assessment framework. *Information and Software Technology*, 137:106600, 2021.

[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-C)*, pages 41–48. IEEE, 2020.

[4] L. Billard and E. Diday. *Clustering Methodology for Symbolic Data*. Wiley Series in Computational Statistics. Wiley, 2019.

[5] G. Bonaccorso. *Machine Learning Algorithms: Popular Algorithms for Data Science and Machine Learning, 2nd Ed.* Packt Publishing, 2018.

[6] Miguel Brito, Jácome Cunha, and João Saraiva. Identification of microservices from monolithic applications through topic modelling. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1409–1418, 2021.

[7] Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley KG Assunção, Juliana Alves Pereira, Baldoino Fonseca, Márcio Ribeiro, Maria Julia de Lima, and Carlos Lucena. On the performance and adoption of search-based microservice identification with tomicroservices. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 569–580. IEEE, 2020.

[8] P. Dangeti. *Statistics for machine learning*. Packt Publishing Ltd, 2017.

[9] P. Di Francesco, P. Lago, and I. Malavolta. Migrating towards microservice architectures: An industrial survey. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 29–38, 2018.

[10] P. Di Francesco, P. Lago, and I. Malavolta. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, 150:77–97, 2019.

[11] M. Hagiwara. *Real-World Natural Language Processing: Practical Applications with Deep Learning*. Manning, 2021.

[12] Anup K Kalia, Jin Xiao, Rahul Krishna, Saurabh Sinha, Maja Vukovic, and Debasish Banerjee. Mono2micro: a practical and effective tool for decomposing monolithic java applications to microservices. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1214–1224, 2021.

[13] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. Towards a technique for extracting microservices from monolithic enterprise systems. *CoRR*, abs/1605.03175, 2016.

[14] Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao, Jidong Ge, and Zhihao Shan. A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, 157:110380, 2019.

[15] Sam Newman. *Monolith to microservices: evolutionary patterns to transform your monolith*. O'Reilly Media, 2019.

[16] A. Selmadji, A.-D. Seriai, H. L. Bouziane, R. O. Mahamane, P. Zaragoza, and C. Dony. From monolithic architecture style to microservice one based on a semi-automatic approach. In *IEEE Intl Conf. on Software Architecture (ICSA)*, pages 157–168, 2020.

[17] Yukun Zhang, Bo Liu, Liyun Dai, Kang Chen, and Xuelian Cao. Automated microservice identification in legacy systems with functional and non-functional metrics. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 135–145. IEEE, 2020.