Automate migration to microservices architecture using Machine Learning techniques

Meryam Chaieb
Laval University
Québec, QC, Canada
meryam.chaieb.1@ulaval.ca

Mohamed Aymen Saied
Laval University
Québec, QC, Canada
mohamed-aymen.saied@ift.ulaval.ca

Abstract

The microservice architectural style has many advantages such as scalability, reusability, and easy maintainability. Microservices have therefore become a popular architectural choice when developing new applications. Reaping these benefits requires redesigning the monolithic application and moving them to a microservices-based architecture. This process is inherently complex and costly, so automating this task is critical.
This report proposes a method by which potential microservices can be identified from a given monolithic application while treating this problem as a clustering thematic.
Our method takes as input the source code of the source application in order to apply different approaches to devise the one box application into its different microservices. In this report we detail each of these techniques while finishing with a discussion and comparison of the results.

Keywords: Machine learning, Monolithic architecture, Microservices architecture, Static analysis of source code, NLP.

Introduction

Software engineering is the science that implements software products to solve real world problems or to facilitate certain tasks. In the last decades, as technology has invaded everyday life, these software products are facing thousands of new users for applications that were previously designed for a limited number of people.

Over the past decades, software engineering research identified and attempted to solve a variety of issues pertaining to several phases of the software lifecycle. However, the fast pace of evolution in the IT industry and the staggering growth of new technologies [50] based on APIs [30, 45, 25], containers [48], microservices [40, 1, 41, 29], cloud and virtualization, put an increasing pressure on software development [2] and deployment [49, 47] practice to fully exploit this paradigm shift. This led to constant questioning of existing techniques [30] and results of software engineering research [36, 35], leading to investigating the use of AI and ML-based techniques to solve software engineering problems in topics related to software reuse [16], recommendation systems [34], mining software repositories [36], software data analytics and patterns mining [38, 19, 37, 31] , program analysis and visualization [39, 33], testing in the cloud environment, Edge-Enabled systems [24], microservices architecture [43] and mobile applications.

We then observed an increased tendency of organizations to move existing enterprise applications to the cloud, also known as cloud computing. This corresponds to the access to IT services (servers, storage, networking, software) through the Internet provided by a provider. There are many reasons for this migration, for example: high availability, automatic scaling, easier management of the infrastructure and compliance with the latest security standards guarantee a more agile and combined development and operation flow.

Driven by this new paradigm, the design, creation, deployment and maintenance of enterprise applications have fundamentally changed. To bridge this gap and make existing monolithic applications ready for injection into cloud technology, they must run as flexible, loosely coupled compositions of specialized services, which has recently become the microservices style of architecture.

In addition, monolithic applications that have been developed over the years can become large, complex and even inefficient. Resulting in obscure structures. On the one hand, this

makes the monolith difficult to maintain with reasonable effort and to adapt to new and better technologies.
On the other hand, they are often unable to scale at the module level, but rather by duplicating instances of the entire application. In most cases, this is an ineffective approach to meeting rapidly changing workloads while maintaining optimal resource utilization.

This is why the software industry invests today in research in the area of migration from monolithic applications recognized as application encapsulated in a single box to the new architectural style of microservices where the application is divided into several boxes each representing a particular service.
Unlike monolithics, microservices applications meet the requirements of the cloud perfectly and their maintenance and development efforts are known to be reasonable.

During this report, we tackle the problem of migrating monolithic applications to the architecture of highly available microservices, taking advantage in this quest of the evolution of research in terms of artificial intelligence. The approach we present in this report helps experts by recommending a way to decompose and it is up to the expert to decide, based on the results of the evaluation metrics and his experience, about the final composition of his microservices.

The work carried out is presented in this report organized as follows:
We start with a first chapter that defines the general context of the project and the different objectives to be achieved as well as the desired results, concluding by presenting the methodology that we adopt for our work. We then move on to the state of the art where we present the various key concepts and related works distilled from the literature and we finish by defining the evaluation metrics of the project. In the third chapter we detail the different approaches we propose to solve this problem. The next chapter takes care of the definition of the different algorithms developed and used to finally conclude with the realization chapter where we expose our realization environment as well as the obtained results ending with a discussion. We end this report with a conclusion while presenting our contributions and our perspectives.

Chapter 1

General overview

Introduction

We begin this first chapter of the report by defining the general context of the project. We then describe the project at a high level while detailing our objectives and expected results, and we finish by explaining the project development methodology.

1.1 Project context

In this section, we define the problematic of our project, the objectives and the expected results.

1.1.1 Objectives

Monolithic applications suffer from a dependency hell where everything is tightly coupled, resulting in long integration times and an inability to track down the source of errors detected during the integration cycle. The scalability of a single application is limited and the system suffers severe downtime during upgrades.
With increasing demands for scalability and frequent maintenance of software products and IT services, companies are adopting a microservices architecture for most of their products to realize many benefits.
Such architectures offer technical flexibility, faster production processes, reusable functionality and even multi-disciplinary development teams.

For these reasons, many companies that previously had products built with a monolithic design have chosen to move their services to a microservices-oriented architecture. However, this has proven to be a time consuming and very expensive process.
The manual migration process forces developers to understand the legacy code base without

proper documentation and break it down into reusable microservices without sacrificing the functionality of the original application.

In this project, we are interested in solving the problems encountered by the practitioner, identified previously, when decomposing the monolithic application into microservices. Our tool will serve as a support for the user by proposing a way to decompose the single box system into multiple services taking into account different criteria that we detail in the following chapter.

1.1.2 Expected Results

At the end of the project, these requirements should be met :

- Collect data from monolithic applications.
- Implement a prototype based on existing tools from state of the art approaches for decomposing a monolithic application to microservices.
- Select a tool to evaluate the decomposition.
- Conduct experiments to evaluate the proposed approach in terms of the collected monolithic applications.
- The code must be well documented and written in a simple and succinct manner.
- The results should be better than the existing results or at least close to them.

1.2 Methodology of work

After having defined the project objectives, project management technique must be implemented to streamline the solution development cycle.
To improve product quality and accelerate the project development process, we adopted the CRISP-DM methodology as our project development methodology which is illustrated in Figure 1.1.
The choice of CRISP-DM as a working methodology comes down to the fact that it is an adaptable process model offering an overview of the data mining life cycle (the sequence of phases is not strictly established).
Our work focused in a first part on data collection and exploration. Then we focused on the modeling and evaluation of the solution. Finally we return to the correction and exploitation of the data so that they represent our problematic at best, which leads us to the desired results.

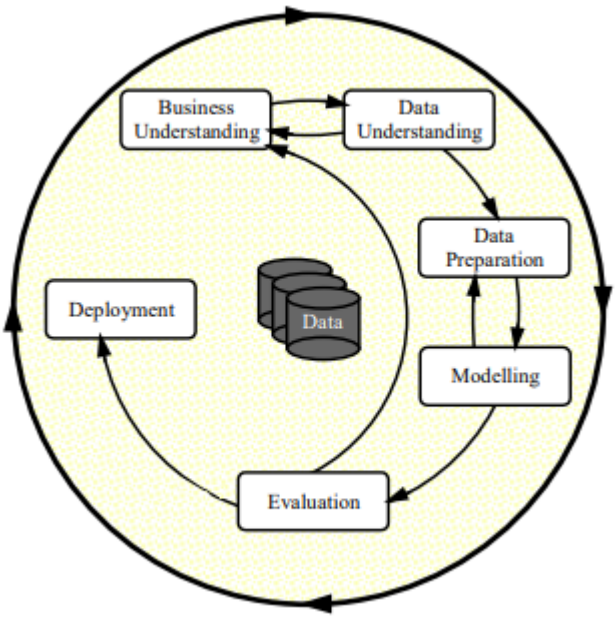

Figure 1.1: CRISP-DM Process Diagram
[51]

The CRoss Industry Standard Process for Data Mining (CRISP-DM) consists mainly of six sequential phases as follows [51]:

- **Business Understanding:** Business Understanding is the first phase of CRISP-DM and focuses on determining the project objectives and requirements. In our case, this step is detailed in the next chapter of the report entitled "State of the Art" where we review key concepts and existing research that has attempted to address similar problems.

- **Data Understanding:** Next comes the data understanding phase. Building on the foundation of business understanding, it focuses on identifying, collecting and analyzing the data sets that can help to achieve the objectives of our project. During this stage, data from the monolithic applications was collected using a static code analysis tool which will be detailed in the following parts of the report. This phase is also illustrated in the third chapter named "Proposed Approaches".

- **Data Preparation:** The necessary data sets are created in this phase. The data goes through a selection, cleaning and reformatting pipeline. This step is also described in the third chapter.

- **Modeling:** At this stage, after choosing the modeling techniques to work with,

their parameters must be adjusted until the best results are obtained. This phase is presented in the "Modeling" chapter.

- **Evaluation:** In this critical step, the results of the models are evaluated to determine the next step in the process. This phase represents an evaluation of the success of the previous phase. In our report, this part is presented during the last chapter named "Realization".
- **Deployment:** Beyond this phase the customer will have access to the software product and to do so a monitoring and maintenance strategy must be created, followed by a final report and a project review. In our case, the code must be clear and concise since it will be used in the future work of the laboratory's research team.

Conclusion

In this first chapter, we presented the general context of the project while detailing the objectives and the expected results. Finally, we have proposed the development process of the project which we adopt in the next chapters.

Chapter 2

State of the art

Introduction

We begin this second chapter of the report by listing the different key concepts leading to the good comprehensibility of the project. Then, we explain the PRISMA research methodology that we used during the selection of scientific articles followed by our literature review. We conclude this first part with a discussion mentioning our scientific contributions to answer the first research question raised in this chapter.
Finally, in a second part of the chapter, we answer a second research question by presenting the different evaluation metrics distilled from the literature.

2.1 Key concepts

For a detailed description of our project, we start by defining the different key aspects of our problem.

2.1.1 Software engineering

Software engineering is the science of industrial engineering that studies the work methodologies and best practices of engineers who develop software. Software engineering applies engineering principles and techniques to the design of software systems such as the definition of the architecture to be respected when creating a project.

In addition to the documentation tasks inherent to the follow-up of any engineering project, the software engineer must be able to perform the tasks of analyzing the needs and requirements of the software product, to design and develop it following professional standards. Also, the software engineer ensures the necessary and adequate tests to guarantee the conformity of its product to the previously defined specifications as he can still maintain his solution [URL7].

One of the famous architectures that has been the support of software product development in the parent companies of the field for years is the monolithic architecture that we define in the following section.

2.1.2 Monolithique architecture

Monolithic architecture was the traditional approach to software development, previously used by large companies like Amazon and Ebay.
In this architectural style, the functions are encapsulated in a single system. In other words, it is a single box that gathers all the functionalities as illustrated by Figure 2.1.

Monolithic applications, as long as they are not complicated, have their own advantages such as ease of use, development, testing and deployment.
However, when the application tends to become more complicated, the monolithic structure expands, becoming a large piece of software that is difficult to manage and scale. The problem is that at present, many companies still have their software as monoliths. The scalability and maintenance problems previously presented force companies to buy new software instead of developing new features in the ones they have[7].
Migrating their monolithic applications to microservices architecture can be a solution and their software will regain its scalability and all the benefits associated with microservices architecture.

2.1.3 Microservice architecture

A new style of architecture called microservice, promises to address the problems of monolithic architecture cited in the previous section. This is a trend in the practice of the software engineering industry that has been defined by Lewis and Fowler [URL8].This architectural style is considered a technique for developing software applications by inheriting the principles and concepts of Service-Oriented Architecture (SOA). It allows structuring an application based on services.
A service is a collection of small, loosely coupled software services as illustrated in Figure 2.1.
Microservices can be seen as a new paradigm for programming applications through the composition of small services, each running its own processes and communicating via lightweight mechanisms.

Microservices are very small entities, as their name indicates, for their contribution to the application and not because of the number of lines of code.
The main characteristics of microservices architectures are limited contexts, flexibility and modularity. Indeed, each entity supports a single service, generally a single use case, and the set of these microservices that constitute the application will thus ensure all the services or functionalities guaranteed by the parent monolithic application. These loosely coupled services can be developed independently.

In this regard, microservices are a new trend in software architecture that emphasizes the development and design of highly maintainable and scalable software [7].

2.1.4 Refactoring of monolithic applications

Refactoring is an activity that aims to extend the life of existing software products. It is then a transformation of the code while guaranteeing the same functionalities in order to improve the source code that has structurally deteriorated with time or has accumulated technical deficiencies.
A lot of research has been done in this area, mainly aiming at refactoring at the source code level. Fowler and al [14] have consolidated the field through their well known books entitled "Improving the design of Existing Code, more than 70 Patterns explained" where they present refactoring techniques based on decomposition patterns. Dietrich in turn calls code-level refactoring as high-impact refactoring [12].

The principle of migrating from a monolithic architecture to a more reliable microservice architecture is to identify contextually related modules and encapsulate them in a single service, while ensuring strong cohesion within a single microservice and weak coupling between microservices.
To get the most out of this refactoring step, choosing the right level of granularity when dividing the functionality is a decisive step. However, building a new application from scratch based on a microservices architecture can be a very expensive and time consuming task.
On the other hand, the process of refactoring a mature monolithic application into microservices also suffers from the same time and cost constraints. It is in this quest for optimization that the industry today invests in research and development in the area of refactoring and application migration [15].

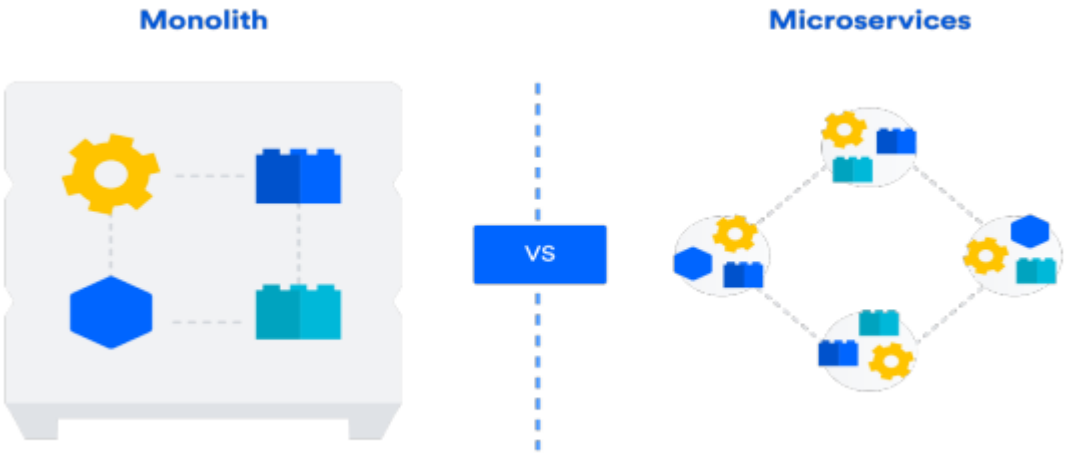

Figure 2.1: Monolithic application Vs Microservices application [URL16]

2.1.5 Machine learning

Machine learning is a form of artificial intelligence, a field that combines computer science and robust data sets to enable problem solving by aiming to copy the decision making process of a human being. The main goal of machine learning is to create systems that learn and improve their performance by examining a stream of data to train and learn models, and then produce descriptions and predictions that ensure optimal decision making, even in the most complex scenarios.[URL11]

2.1.6 Natural language processing

Natural language processing is one of the most dynamic research areas in the field of data science. It is a field at the intersection of machine learning and linguistics. Its goal is to extract information and meaning from textual content.
Among the text processing techniques that we exploit in our project, we cite [23] :

- **Empty word removal :** Empty word removal is one of the basic preprocessing operations in various automatic language processing applications. The principle consists in eliminating words that appear in the commonly in all the documents of the corpus.
  An empty word can be defined as a word that does not add meaning to the text and/or its deletion does not change the meaning of the sentence.
  Generally speaking, articles and pronouns are usually classified as empty words.

- **CamelCase :** This is a writing standard commonly used in programming languages. Indeed in the names of variables or classes the words are not separated by spaces but rather to indicate the beginning of the second word we mark it with a capital letter. Therefore, we use this same principle to separate words in the document. For example the word CamelCase will be : [Camel, Case].

- **Stemming Vs Lemmatization :** In linguistics, stemming or desuffixation is a process of reducing words to their root.
  The root of the word is the part left after we proceed by deleting its prefix(es) and suffix(es).
  Unlike the lemma, which corresponds to a term taken from the ordinary usage of the speakers of the language, the root generally corresponds only to a term derived from this kind of analysis as illustrated by the attached figure 2.2.

Figure 2.2: Stemming Vs Lemmatization

2.1.7 Term Frequency Inverse Document Frequency

TFIDF is a technique used to vectorize data. It does this by proportionally increasing the number of times a word appears in the document, but it is counterbalanced by the number of documents in which it is present. Therefore, words that are commonly present in all documents, do not have a very high rank. However, a word that is present too often in a small number of documents will be assigned a higher rank because it may be indicative of the context of the document.[22]
We use this technique to vectorize our data and we'll detail it in the following chapters.

2.1.8 Clustering

Clustering is essentially a type of unsupervised machine learning method, based only on input data that is unlabeled. In other words, clustering is a process for finding meaningful structure, generative features and inherent groupings in a set of examples.
Clustering algorithms partition the data into a number of clusters (groups or categories). A cluster is described by considering internal homogeneity and external separation. That is examples in the same cluster should be similar to each other (thus increasing cohesion within the cluster), while those in different clusters should not (thus reducing the range between clusters) [53].

Density-based clustering is a non-parametric approach where clustering methods do not require the number of clusters as input parameters. In this type of clustering, a cluster is a set of example data distributed in the data space over a contiguous region of high example density.
However, density-based clusters are separated from each other by contiguous regions of low object density. Thus, data objects located in the low density regions are generally considered noise or outliers [5].

As mentioned, we consider this refactoring problem as a clustering problem in which we have opted for different clustering algorithms especially density-based ones.

2.1.9 The graphs

A graph is a non-linear data structure used to represent real world problems. A graph consists of a finite set of nodes (vertices or points) connected to each other by edges. These nodes can be in an arbitrary form forming an undirected graph or in the form of a set of oriented pairs for a directed graph.

This type of data structure emphasizes the relationship between objects as well as between the nodes of the graph. Therefore, the edges of the graph may contain a weight reflecting the importance of the relationship or link between these two nodes. [13]
A main application of graphs is the detection of communities that we define in the following lines.

A community can be defined as a subset of nodes that are loosely connected to each other and loosely connected to nodes in other communities in the same graph. Detecting communities in a network is one of the most important tasks in network analysis. In a large scale network, we will need community detection algorithms that can partition the network into multiple communities[URL4].
There are mainly two types of methods for detecting communities in graphs :

1. **Agglomerative methods:** In this approach, we start with an empty graph composed only of the nodes of the original graph. Then, edges are added one by one to the graph, starting with the edges of the highest weight (depending on the application) [URL4].

2. **Splitting methods :** In this method, we proceed in the opposite direction. We start with the complete graph and remove the edges iteratively. The edge with the lowest weight is removed first. After a number of steps, we obtain groups of densely connected nodes [URL4].

In this project, we use graphs to represent our problem. Then, we apply community detection algorithms in order to detect microservices.

The introduction of these definitions of key concepts, which we use in the development of our project, allows us to move on to the second part of the chapter through the first research question.

Research Question 1 : What refactoring approaches exist in the context of decomposing a monolithic application architecture into its microservices and how can they be classified based on the techniques used?

2.2 Related work

After defining the key concepts and to answer our first research question presented above, in this section we start by explaining the research methodology we have used, then we present the different existing solutions classified according to the approach used and we end with a discussion.

2.2.1 Research Methodology

During our research process we followed the PRISMA approach which we present in the next section because it is simple to apply and successful in conducting literature reviews.

2.2.1.1 Presentation of PRISMA

PRISMA is an abbreviation for Preferred Reporting Items for Systematic Reviews and Meta Analysis. The process consists of a set of evidence based reporting items for systematic reviews and meta-analyses. PRISMA is primarily designed to report on reviews that assess the effects of interventions, although it can also be used to describe systematic reviews with objectives other than the evaluation of interventions.[27]

2.2.1.2 Application of the PRISMA method

As shown in Figure 2.3 the steps in the PRISMA process are :

- **Identification phase :** This is the first step in the PRISMA process and it reintroduces the step of collecting related work based on keywords that match the work. We searched for research articles using a combination of a set of keywords on Google scholar [URL6] as a search engine, since it allows users to access the entire scientific literature from a single location.
  Connected papers [URL2] was also a useful tool as it helped us to select the articles connected to our research theme by inserting a first article representing our research interest and generating all the articles semantically connected to it.
  The date range was from 2014 to 2022 using the key phrases: "Monolith refactoring", "Monolith migration", Monolith to microservices", "GNN for Monolith decomposition", "Machine Learning for monolith refactoring", "Microservices based application", "Graph community detection".

  For this step, we encountered 44 items. Five of them were deleted because they were duplicate items.
  In this first phase, despite the limited time frame of the project, we have tried to select the most relevant research articles from among the most recent ones. Even if we can assume that this work is not exhaustive, the selected research works give a clear overview of the approaches already developed to solve this problem.

- **Selection phase :** This is the filtering stage of the selected articles. The first selection criterion is based on the title and the abstract, depending on whether or not they correspond to the project. The second criterion is the availability of the full text of the article.
  In the end, a complete reading of the articles is done to evaluate their eligibility.
  In our situation :

  - 8 items were excluded based on the Title/Abstract sort.
  - 6 were not recovered because we were unable to find their full texts.
  - 5 reports were excluded after review of their full texts because the approaches used were not clearly explained.

- **Inclusion Phase :** This is the reading and vetting stage of the articles that have met all of the previously mentioned screening criteria.
  In our case, 20 articles were included in the review. On the one hand, 11 of them contain generalities on the key concepts and the field of application. On the other hand, the other 9 articles present recent approaches dealing with similar issues. The latter nine research articles are used in the following comparative study.

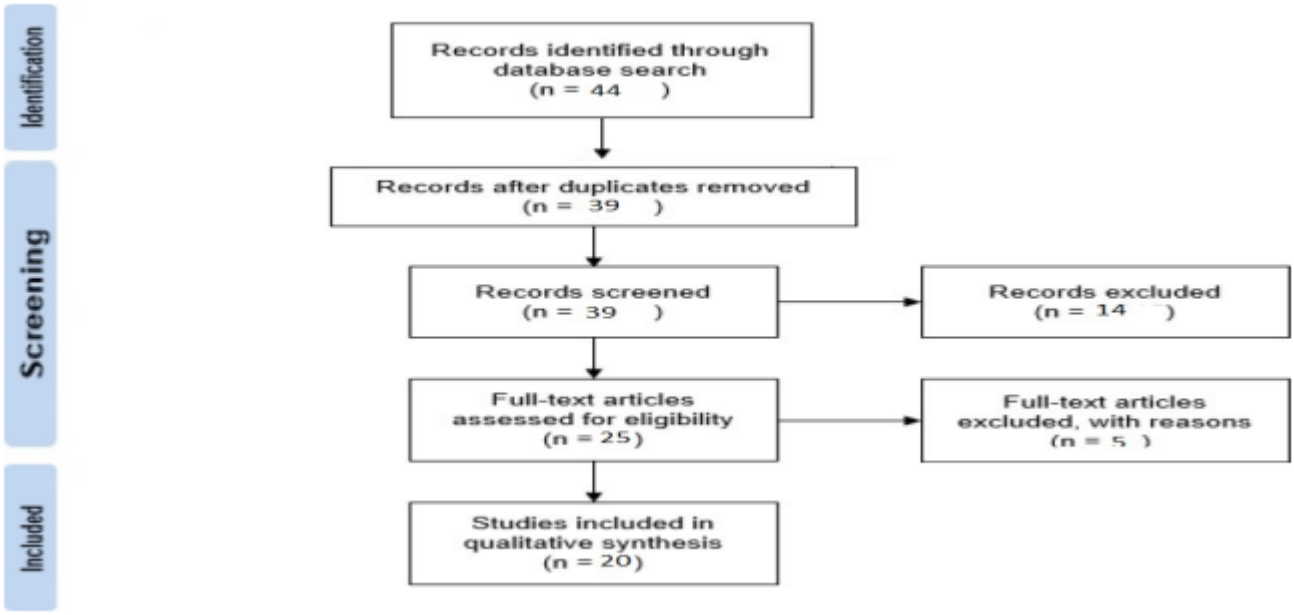

Figure 2.3: PRISMA-inspired flowchart for article retrieva
[18]

2.2.2 Existing approaches

In this section, we present the work done using the different articles selected through the PRISMA filters.
For that, we divide them according to the types of analysis that they carry out during the realization of their solutions. Finally, we examine the extent to which these methods were able to solve the project problem and conclude with our contributions.

2.2.2.1 UML diagrams

The Unified Modeling Language (UML) is a graphical design language based on diagrams and is designed as a standardized modeling method for software development and object oriented design [21].

Service Cutter is a very special migration tool as it is the only one in the literature that is based only on UML diagrams to represent the different aspects to be treated of the monolithic application to be decomposed. Starting from these diagrams, which are rewritten in json format and provided by the user, Service Cutter generates a weighted undirected graph representative of the application whose nodes are nanontities. The latter are operations or methods and the edges are the interactions that take place. This graph is generated based on 16 coupling criteria that Gysel and al.[17] stated distilled from industry experience. This decomposition tool also allows the expert user to order his

criteria according to his preferences. In a final stage of the microservices recommendation process, Gysel et al. apply Girvan Newman's community detection algorithm and finally generate a decomposition recommendation by defining the services and the nano-entities that belong to it. It should be noted that this approach works on the basis of 9 software diagrams that touch several aspects. Its 9 diagrams are : Entity-relationship diagram, Use cases diagram, Shared owner groups, Aggregates, Entities, Predefined services, Separated security zones, Security access groups, Compatibilities.

The problem with this approach is that most of the applications that exist are missing representative diagrams and therefore it is up to the user to use reverse engineering techniques to generate them and then transform them into the proper format accepted by this approach. In addition, in terms of comparison of results, the developers of the Service Cutter tool do not use known metrics in migration literature but rather rely on comparison with a manual decomposition process that relies on domain experts.

Nevertheless, Dehghan and al.[9] have based their recently published approach on the Service Cutter tool described above. Indeed, the user provides the necessary models and the source code to two independent mechanisms:
On the one hand, the Service Cutter tool receives the different diagrams in order to generate a recommendation of the microservices-nanoentities.
On the other hand, MoDisco (Model Driven Reverse Engineering Framework) [4] receives in turn the Entity-Relational model and the source code to generate a recommendation of the corresponding methods-nanoentities. This approach is based on reinforcement learning where the goal is to associate each method to its microservice according to the pre-results previously synthesized by the two separate tools.

Like the Service Cutter, the verification of the quality of the results provided by this approach was based on a manual process since the applications they used are small. It should also be noted that for large systems, the system takes considerable time to stabilize.

2.2.2.2 Static analysis of the code

In computer science, the notion of static program analysis encompasses a range of techniques for collecting information about the behavior of a software program during its execution, without having to execute it.

In this quest Brito and al.[3] have developed a somewhat particular approach where the source code is considered as a text to be processed from which they have extracted information in two forms:
The first part is lexical by detecting all significant words in the source code and structuring them with an abstract syntax tree (ASA) and the second part is structural by using JavaParser Symbol Solver to identify the type of each given expression. The lexical data are used as input to the topic modeling mechanism which combines its output with that of

the structural analysis to feed the clustering phase that generates the final clusters. The results of this approach were found to be acceptable.

In the other end of the approaches based on static analysis of the code we present the solution entitled "HierDecomp". We were inspired by the article of this last approach to develop our third research track that we detail in the following chapters.
This article published by Sellami and al.[42] is based on the treatment of the problem from a clustering point of view. According to their approach on two kinds of metrics that they defined as the structural similarity synthesized from the static calls between the classes of the application and the semantic similarity derived from the analysis of the code text (we detail these two metrics in the third chapter of our report when presenting our third approach). Candidate microservices are generated following a hierarchical clustering done on the basis of these last metrics. Sellami et al have shown good results when compared to several distilled approaches in the literature.

2.2.2.3 Dynamic code analysis

Dynamic computer product analysis is a form of program analysis that requires the execution of programs. It is intended to study the behavior of software and the results of its execution on its environment.[6]
Monolithic application migration approaches that rely on dynamic source code analysis have provided better results.

Among which we mention the proposal of the IBM research team in 2021, Desai and al [11], who proposed an approach in which they take advantage of the advancement of research in terms of GNN techniques. It is a method that treats the problem as a class-level clustring

They start by generating an unweighted directed graph representative of the application whose nodes represent the classes and edges represent a relationship between them. The vectors that encode the nodes of the graph are synthesized thanks to the information extracted from the application's execution traces.

The biggest advantage of this approach is that at the end of the process the system not only generates candidate microservices but also shows the user which classes in each cluster have a high interaction rate with the rest of the classes. According to the literature, this type of class requires a lot of attention during refactoring. The results of this GNN based approach outperform the rest of the graph-based methods but to take advantage of this solution the user has to provide the number of clusters as input which is not a simple task.

However, and in the same context, IBM has recently published a refactoring tool called "Mono2Micro", which is known to be the most efficient in the industry. Kalia and al. [20] have defined 6 steps for this migration process. They start with the collection of execution

traces due to the source code provided by the user. Then, for each use case, they proceed to the reduction of these traces: First, by considering only unique traces. Second, by reducing the length of a trace by removing redundant sequences of classes that could have been invoked due to the presence of a loop.
After a series of processing, the team ends up applying a hierarchical clustering to extract the microservices and the user has access to the different levels of clustering.
IBM also offers an interactive interface through which the user can visualize the final result and can associate a class with another microservice of his choice different from the one assigned to him by the decomposition tool and continue to compare the results he obtains.
Mono2Micro is the excellent decomposition tool in the monolithic application migration industry.

2.2.2.4 Decomposition patterns

Sam Newman being an expert in the field at the industrial research level defines in his book entitled “ "Monolith to Microservices Evolutionary Patterns to Transform Your Monolith “ the different patterns that help in the decomposition of monolithic applications[26].
The most famous of these patterns is called the "Strangler Pattern" or "The Fig Principle". This strangler pattern was inspired by a certain type of fig that is planted on the upper branches of trees. The fig then descends to the ground to take root, gradually enveloping the original tree. The existing tree initially becomes a support structure for the new fig. In the final stage, we can observe the original tree die and rot, leaving only the new, now self sustaining fig in its place.

In the context of software, the parallel principle is that our new microservices-based system is initially supported by the existing system and the wrapper.
The idea is to have the old and the new coexist, giving the new system time to develop and fully replace the old system. The key advantage of this model is that it supports the goal of enabling an incremental migration to a new microservices-based system. In addition, it provides the ability to pause and even stop the migration completely, while continuing to take advantage of the new system delivered so far.

As shown in figure 2.4, the process is simple: We start by designating the functionality to be extracted (to copy the code if possible) from the running monolithic application, which corresponds to the first step in the figure. During the extraction of the functionality, the calls are always directed to the parent monolithic application to ensure that the services are not broken during this phase, this is illustrated by the second step of the process. After the extraction phase is over and the functionality has become a full-fledged functional service, the calls concerning it are then directed to the new microservice that represents it and so

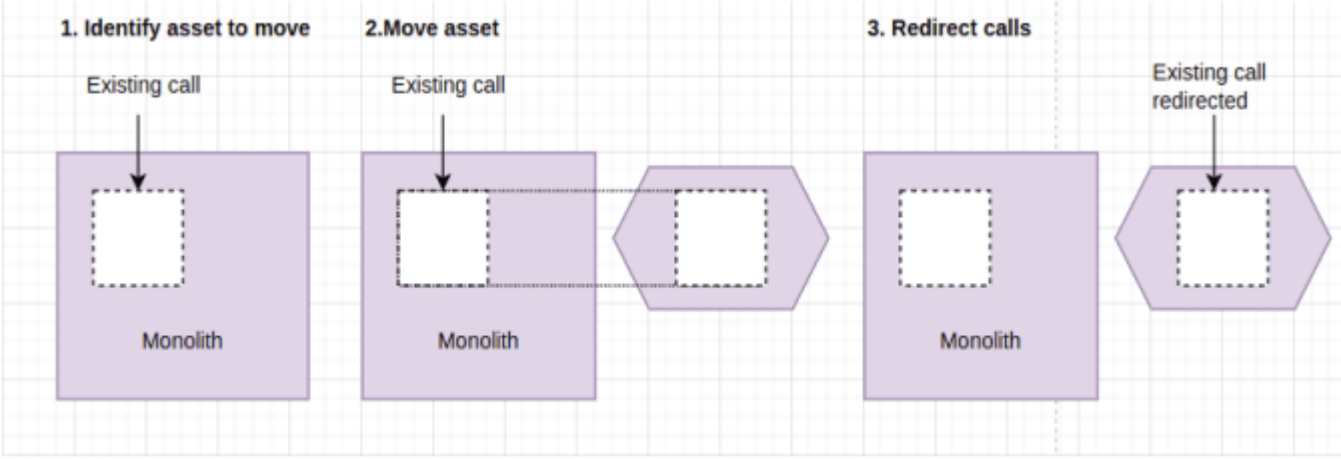

Figure 2.4: Illustration of the strangler pattern
[26]

on. This is represented through the last step in our figure 2.4.
The end of our process is marked by the degeneration of the parent monolithic application.

2.2.2.5 Discussion

There has been considerable interest in service-oriented or microservice architectures that allow for rapid system modification. Refactoring and, in particular, remodularization operations can be performed to repair the design of a software system. Various approaches have been proposed to help developers when remodularizing a software system. The common problem in these efforts is to identify from monolithic applications the candidates for microservices, i.e., programs that can be transformed into cohesive, self-contained services. There are various decomposition approaches that in this section we have tried to gather according to the type of analysis done by presenting their general aspects.
In order to take advantage of these approaches to build our own contribution, we need to define an evaluation tool and this is what allowed us to pose our second research question.

Research Question 2 : In view of the different approaches extracted from the literature that are the subject of solutions for our problem, How can we evaluate these solutions and decide on their robustness ?

2.3 Evaluation metrics

In this section and in order to answer the second research question we have raised, we present the evaluation metrics we use in the evaluation process of approaches including ours so that we can position ourselves in this research quest. These metrics are distilled from the

literature mainly from the research work of IBM, the industry and research leader in the field of monolithic application migration through its famous paper Mono2Micro [20] dating back to 2021 and whose approach we detailed in section 2.2.2 during the presentation of existing approaches.
The metrics we consider are:

- **Structural Modularity (SM) :** Structural modularity measures the quality of partition modularity as the structural cohesion of classes within a partition m_i (scoh) and the coupling (scop) between partitions(M), represented by equation 2.1:

$$\frac{1}{M}\sum_{i=1}^{M} scoh_i - \frac{1}{(M(M-1))/2} scop_{ij} \tag{2.1}$$

  With :

  - $scoh_i = \frac{\mu_i}{m_i^2}$,
  - μ_i represents the number of calls within the partition m_i,
  - $scop_{ij} = \frac{(\gamma_{ij})}{2*(m_i*m_j)}$,
  - γ_{ij}represents the number of calls between partitions m_i and m_j.

  → The higher the SM value is, the better the recommendation..

- **ICP:** Measures the percentage of calls occurring between two partitions, represented by Equation 2.2 :

$$icp_{ij} = \frac{c_{ij}}{\sum_{i,j=0j/=i}^{M} c_{ij}} \tag{2.2}$$

  with :

  - c_{ij} represents the number of calls detected between partition i and j.

  → The lower the ICP value is, the better the recommendation.

- **Interface Number (IFN) :** Measures the number o f interfaces in a microservice. An interface is defined as a class of a microservice m_i that is called by another class of another microservice m_j. This metric can be calculated using equation 2.3 :

$$\frac{1}{N}\sum_{i=1}^{N} ifn_i \tag{2.3}$$

  With :

  - N being the total number of microservices,

– $if n_i$ represents the number of interface classes in the microservice m_i.

→ The lower the IFN value is, the better the recommendation.

- **Non-Extreme Distribution (NED) :** This metric reflects the way classes are distributed within microservices. It is preferable that a microservice be neither very large nor very small in terms of the number of classes it contains. Wu and al.[52] define a microservice as non-extreme if its size varies between [5 , 20] classes. This value is measured by equation 2.4 :

$$1 - \frac{\sum_{k=0}^{N} n_k}{|N|} \tag{2.4}$$

With :

– n_k represents the microservice classified as non-extreme,

– N being the total number of microservices.

→ The lower the NED value, the better the recommendation.

- **DUP :** This is another metric that controls the size of microservices by calculating the number of duplicated classes. Based on the fact that one of the solutions to reduce the coupling between partitions is the duplication of the co-used classes.

→ The lower the DUP value is, the better the recommendation.

Conclusion

In this chapter, we started by enumerating the key concepts necessary for a good understanding of our project. Then, in a second part, we presented the PRISMA research methodology that we followed when selecting the scientific articles that we exploited in the existing approaches part, followed by a discussion in order to answer the first research question elaborated previously.
Finally, and to answer the second research question, we have closed this chapter by detailing the evaluation metrics used to compare and position the approaches, including our own, which we will present in the following chapters.

Chapter 3

Proposed approaches

Introduction

In this third chapter of the report, we start by presenting the data understanding phase in a first section while mentioning the tools used to satisfy this task. Then, in a second section, we detail the proposed approaches by focusing on the data preparation part, the second phase of the CRISP-DM process.

3.1 Data understanding

We treat this refactoring problem as a clustering problem. Our solution takes as input the source code of the applications to be split into microservices. The level of granularity we consider is at the class level.

A class : A class in the Java object-oriented programming language is a basic element that can be introduced as a definition model for objects with the same set of attributes, and the same set of operations. A class allows to create one or more objects by instantiating. Each object is an instance of a single class [URL3].

Our solution does not replace the decision of the expert but recommends a way to gather the classes of the application while presenting the different results of the evaluation metrics that are suitable to evaluate the approaches to solve this migration problem and that we presented in the previous chapter in section 2.3.

To do so, we collected two monolithic applications to train our data and test our clustering approach.

Indeed, these applications have been chosen according to the following criteria :

- Monolithic application written in Java.

- Availability of source code.
- To be used by other approaches so that we can compare approaches performances to the results of different reference approaches.

However, the monolithic applications that form our database are:

- **Daytrader [URL12]:** This sample contains the DayTrader 7 benchmark, which is an application built around the concept of a stock trading system online. The application allows customers to log in, view their portfolios, check stock prices and buy or sell shares.
- **Acmeair [URL1] :** This application presents the implementation of a fictitious airline called "Acme Air". The application was built with some key business requirements in mind: the ability to scale to billions of calls of Web APIs per day, the need to develop and deploy the application in public clouds (as opposed to a dedicated pre-assigned infrastructure), and the need to support multiple user interaction channels.

We summarize in this table the characteristics of the applications mentioned above:

Project	Version	Number of lines in the code	Nbr of classes
Acmeair [URL1]	1.2	8,899	86
DayTrader [URL9]	1.4	18,224	118

Table 3.1: Characteristics of monolithic applications

After selecting the monolithic applications and retrieving their source codes, we start the data collection process. Indeed, our approach is based on the calls between the classes of the application. We perform a static analysis of the source code. To do this, we use a static code analysis software called "Understand".

Understand

Understand, one of SciTools' products, is a customizable integrated development environment (IDE) that enables static code analysis through a set of visual, documentation and measurement tools. It is designed to make it easy for software developers to understand, maintain and document source code. It enables code understanding by providing flowcharts of relationships and building a dictionary of variables and procedures from a provided source code [URL14].

In our project, during the data collection phase, we use the API provided with Understand to generate the class dependency matrices for each of the applications by considering only the calls between classes.
We detail the generated dependency matrices in the next part of our chapter.

3.2 Research avenues explored

During the development of our problem solving approach, we conducted three rounds of the CRISP-DM work methodology mentioned in the first chapter of our report. As a result, we explored three avenues of research discussed in this section.

3.2.1 First approach : Naive method

In this first approach, we start by generating the dependency matrix between the classes of the "Acmeair" application as shown in figure 3.1 :

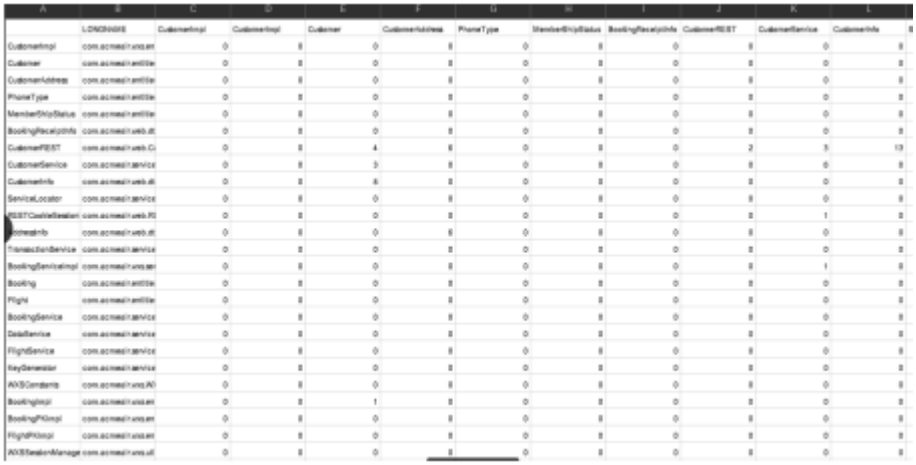

Figure 3.1: Extract of the dependency matrix between the classes of the Acmeair application

From this matrix, we generated a tuple to represent each class of the application in question in the form [a , b] such that :

$$a = \sum call_{in} \tag{3.1}$$

$$b = \sum call_{out} \tag{3.2}$$

With :

- $call_{in}$ represents the incoming calls from all other classes to the class in question,

- $call_{out}$ represents outgoing calls from the class in question to all other classes in the project.

In the end, the data we process in this first approach are only a representation of each class of the application formed using equations (3.1) and (3.2).

The idea behind this choice of representation is indeed due to the observation that in the dependency matrices, we find classes that interact a lot in terms of outgoing calls but that these same classes are also called several times by other classes. In other words, after this observation, we want the classes that interact a lot to be in the same cluster in order to reduce the coupling between the clusters and to increase the cohesion.

3.2.2 Second approach: Codependent call method

In this second approach, we start by generating the matrix of dependencies between the classes illustrated in the previous figure 3.1 but this time we refine the metric considered when generating the class representations.
Indeed, in this approach we consider codependent calls between classes. To make this notion clear, we proceed by the example of the attached figure 3.2 :

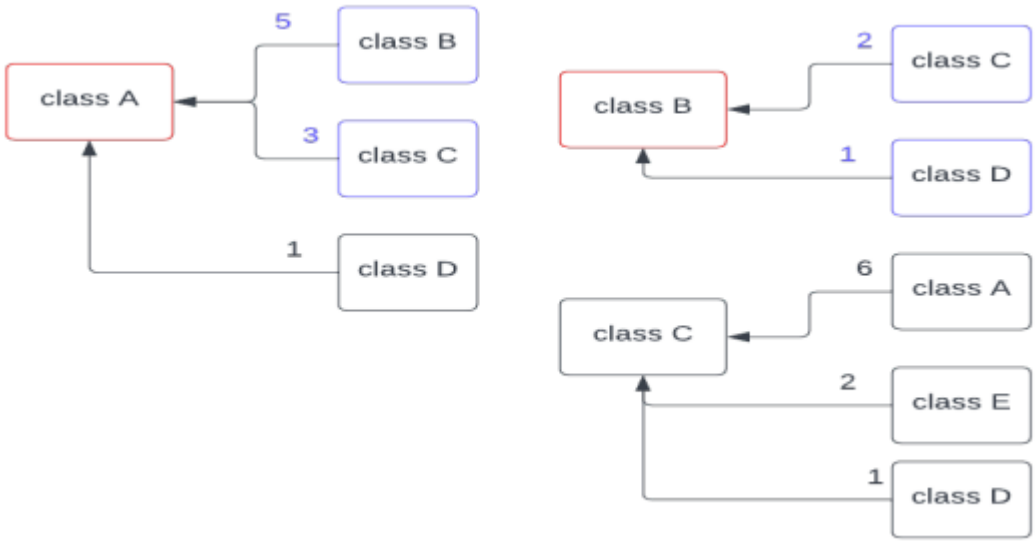

Figure 3.2: Illustrative example of the metric considered

Considering five classes A, B, C, D, and E. Our goal in this example is to encode class A.

The figure shows that class A is called 5 times by class B, 3 times by class C and a single time by class D.

We can notice that the classes that commonly call classes A and B are classes C and D. Also the classes A and C have in common a single class D. Therefore, class A is encoded according to class B (i.e. according to the class that has a maximum of incoming calls from the same classes as those calling class A).

In other words, we consider a tuple representing the class A in the form [a , b] such that :

$$a = \sum call_i \tag{3.3}$$

$$b = \frac{call_{in}A}{call_{in}B} \tag{3.4}$$

With :

- $call_i$: : represents the calls of codependent classes between A and B,
- $call_{in}A$: is the set of all incoming calls to A.
- $call_{in}B$: is the set of all incoming calls to B.

Finally, the line that will represent the class A is none other than the combination of the two equations (3.3) and (3.4).

This representation metric was inspired by the article "Mining Multi-level API Usage Patterns" [32] dealing with the theme of API usage pattern extraction. We are inspired by this same metric used for our second research track for our monolithic application migration problem.

The idea behind this choice of representation is to gather the classes that are called together in the same microservice and this will guarantee not only a weak coupling between the resulting microservices but also the criterion of only one use case invoked by microservice.

3.2.3 Third approach : Graph method

During this last approach, we based on the research paper entitled "A Hierarchical DBSCAN Method for Extracting Microservices from Monolithic Applications" [42] dealing with the theme of our project
In this approach, we no longer encode each class separately but rather the relationship or link between pairs of classes.

For the purpose of extracting relationships and dependencies between different classes, we define two distinct types of similarity between classes as follows :

- **Structural similarity (sim_{str})** : This metric is based on the shared number of method calls between 2 classes. It encodes the dependency between them and thus evaluates the similarity from a functional point of view. The objective of the grouping of classes based on this similarity is to obtain consistent microservices from an implementation point of view.

  Considering two given classes C_i and C_j, the structural similarity is defined by equation (3.5) :

$$sim_{str}(c_i, c_j) = \left\{ \begin{array}{ll} \frac{1}{2}(\frac{call(c_i, c_j)}{call_{in}(c_j)} + \frac{call(c_i, c_j)}{call_{in}(c_i)}) & if \;\; call_{in}(c_i) \neq 0 \;\; and \;\; call_{in}(c_j) \neq 0 \\ \frac{call(c_i, c_j)}{call_{in}(c_j)} & if \;\; call_{in}(c_i) = 0 \;\; and \;\; call_{in}(c_j) \neq 0 \\ \frac{call(c_i, c_j)}{call_{in}(c_i)} & if \;\; call_{in}(c_i) \neq 0 \;\; and \;\; call_{in}(c_j) = 0 \end{array} \right\} \tag{3.5}$$

  With:

  - call(c_i,c_j) : represents the number of times a method of class c_i i has called a method of class c_j.
  - $call_{in}(c_i)$: represents the number of incoming calls in c_i.

  The values of sim_{str} (c_i,c_j) are in the interval [0,1] où 1 where 1 indicates that the classes c_i and c_j j are very similar and used exclusively together and 0 indicates that they are completely independent.
  To calculate this metric, we used the dependency matrices generated by Understand.

- **Semantic Similarity (Sim_{sem}) :** This metric uses natural language processing (NLP) to measure the degree of relatedness of the domain semantics of two given classes. Given that a microservice must provide a specific and/or a single use case in the domain, we need to identify classes that serve similar use cases.
  Assuming that monolithic projects have been coded using standard practices where class, method and variable names reflect functional concepts and comments describe their function. The terminology used within these components can be a powerful tool for extracting the domain meaning of each class.
  Thus, each class is defined by the set of words contained in its comments, parameter names, method names and variable names as illustrated in figure 3.3 where the first

column represents the name of the class and the last column encloses the list of extracted words :

	simple_name	long_name	words
0	CustomerImpl	com.acmeair.wxs.entities.CustomerImpl	[custom, impl, serial, version, uid, id, passw...
1	Customer	com.acmeair.entities.Customer	[custom, get, custom, id, get, usernam, set, u...
2	CustomerAddress	com.acmeair.entities.CustomerAddress	[custom, address, get, street, address1, set, ...
3	PhoneType	com.acmeair.entities.Customer.PhoneType	[phone, type, unknown, home, busi, mobil, valu...

Figure 3.3: Extract from the semantic analysis of the Acmeair project

Each word is pre-processed using natural language processing techniques including splitting using CamelCase, filtering out unnecessary words and stemming.
The final result is a vector of size n_v where n_v is the number of words in the domain extracted from the monolithic application. Finally, this vector is measured using a TF-IDF (Term Frequency-Inverse Document Frequency) model.
The semantic similarity metric is defined as the cosine similarity between two classes, as presented by equation (3.9) :

$$Sim_{sem}(c_i; c_j) = \frac{\vec{c_i}.\vec{c_j}}{||\vec{c_i}||.||\vec{c_j}||} \tag{3.6}$$

With:

- $\vec{c_i}$ and $\vec{c_j}$: represent the TF-IDF vectors of class c_i and class c_j.
- $||\vec{c_i}||$: represents the Euclidean norm of the vector $\vec{c_i}$.

The values of Sim_{sem} are between 0 and 1, where the value 1 means that both classes use exactly the same vocabulary and therefore serve the same use case.

- **Class Similarity (CS) :** The previous similarity metrics represent different aspects of the relationships between classes. These two metrics are not necessarily correlated and using only one of them does not guarantee the satisfaction of the other. For these reasons, we use the class similarity metric which represents a weighted sum between the structural similarity and the semantic similarity of two given classes presented by equation (3.10) :

$$CS(c_i, c_j) = \alpha Sim_{str}(c_i, c_j) + \beta Sim_{sem}(c_i, c_j) \tag{3.7}$$

With :

- $\alpha \in [0,1]$,
- $\beta \in [0,1]$,
- $\alpha + \beta = 1$.

Conclusion

In this chapter we have started by presenting the data understanding phase corresponding to our project by defining the level of granularity considered when solving the migration problem as well as shedding light on the monolithic applications that we consider for the treatment of the problem.
Then, after defining that our approach is classified under the section of solutions dealing with the problem using static analysis of the code, we highlighted the static analysis tool that we used during our research.
Finally, and during the second part of this chapter, we swept through the three research avenues we explored while highlighting the data preparation phase of the CRISP-DM methodology.

Chapter 4

Modelling

Introduction

After having collected and pre-processed the data for each of the approaches explored in the previous chapter, we focus in this part of the report on the modeling phase of the CRISP-DM process. To do so, we detail in this fourth chapter the algorithms developed for each of the three approaches.

4.1 Modeling the naive approach:

On the basis of the data that we select and reserve for this first solution and as we treat this problem from a clustering point of view we have thus applied two clustering algorithms on this last input.
The two chosen algorithms are DBSCAN and BMSC. In the following section, we detail each of these two algorithms.

4.1.1 Density-Based Spatial Clustering of Applications with Noise

The first algorithm used is Density-Based Spatial Clustering of Applications with Noise (DBSCAN).

4.1.1.1 Generalities and benefits

DBSCAN is a density-based clustering algorithm used to cluster data of any shape in the presence of outliers and noise in a large amount of data.[46]
As shown in figure 4.1, the main advantage of the DBSCAN algorithm over partitioning and hierarchical clustering algorithms are:

1. The DBSCAN algorithm can be applied to any arbitrary shape of data rather than spherical or convex shapes. This makes it much more practical than k-means and other clustering algorithms.

2. It has more specific advantages over the k-means algorithm, such as not having to specify the value of k which is the number of clusters, so the number of clusters can be arbitrary depending on the data which makes the clusters larger and more accurate.

3. The major advantage of this algorithm, as its name suggests, is that it is also effective in the presence of noise and outliers.

   Figure 4.1 shows the difference in clustering results between DBSCAN and Kmeans on different shaped datasets where each color in the figure represents a cluster.

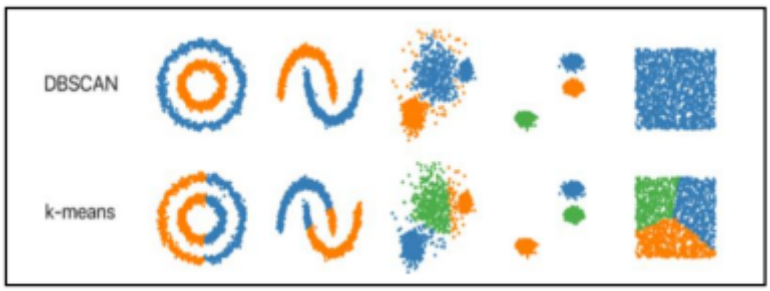

Figure 4.1: DBSCAN vs K-means [46]

4.1.1.2 Hyperparameters

DBSCAN is designed to discover clusters and noise in a spatial database. The researchers need to know the parameters Eps and MinPts . After that, it gathers the densely connected points together in a single cluster using the parameters already entered.
DBSCAN forms the cluster using 2 parameters Eps and MinPts:

- **Eps (ϵ) :** This is the neighborhood radius around the point selected as the center of the cluster to be formed. It is therefore used to determine the densest region.
  → If a large Eps value is chosen, DBSCAN will form less dense clusters and thus the number of detected clusters will decrease.

- **MinPts :** This is the minimum number of points required to form a cluster.
  → The MinPts parameter should not be very low.

4.1.1.3 Types of detected points

After the completion of the DBSCAN algorithm on any dataset we get mainly 3 types of points as shown in figure 4.2 :

- **Core point :** : This is the point from which there are at least a number "k" of points (MinPts) at a distance "r" of radius (Eps).
- **Border point :** This is any point that has one or more center points at a radius "r" (Eps).
- **Noise :** Any point that is neither a core (center point or core) nor an edge point. In our project, we do not take into account the noise points detected by DBSCAN.

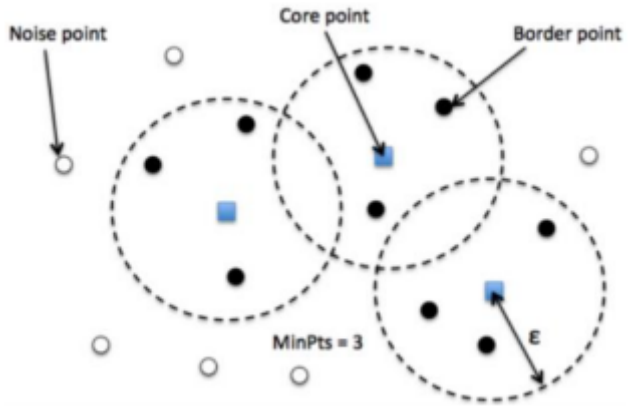

Figure 4.2: DBSCAN [46]

4.1.1.4 Application steps

Finally, the steps to realize the DBSCAN algorithm are as follows:

1. Select an arbitrary point in the database and consider it as our first core point.
2. The collection of data points that are at a maximum distance equal to Eps
3. If the number of points collected is greater than or equal to the minimum number of points required (MinPts), the cluster will be formed.
4. To expand the cluster, these same steps will be repeated for each of the cluster points.
5. At this stage the algorithm generates the first cluster. It will then delete all the points that build it from the database and start the same process again. The algorithm stops when no more clusters can be formed considering the parameters already provided. The remaining points will be classified as noise.

4.1.1.5 Limites of DBSCAN

Despite the good results of the DBSCAN algorithm, it has some limitations such as :

- It has been observed that its performance is poor on variable density databases. Indeed, the Eps parameter always reduces the selection of very distant points and thus points with variable density which does not satisfy the Minpts and therefore DBSCAN creates wrong clusters.
- It is highly user dependent as it takes 2 input values, namely Eps and MinPts. In general, the user does not have a deep knowledge of the data which leads to erroneous clusters.
- It causes some computational overhead that is not always appropriate for large datasets.

To conclude, after having obtained the experimental results that we present in the fifth chapter of the report, we develop a second algorithm based on a theoretical paper entitled "Boosted Mean Shift Clustering (BMSC)" known to be more robust than DBSCAN. In the following section, we detail the BMSC algorithm.

4.1.2 Boosted Mean Shift Clustering

BMSC is a clustering algorithm that we implement on the basis of its theoretical paper. This last one comes to merge two clustering algorithms which are Mean shift and DBSCAN that we have just explained its principle.
Indeed, Mean Shift is a non-parametric clustering technique that does not require the number of input clusters and can find clusters of arbitrary shapes. Although attractive, the performance of the Mean shift algorithm is sensitive to the selection of its parameters. DBSCAN is an efficient density-based clustering algorithm, but it is also sensitive to its parameters and usually merges overlapping clusters. In this section, we propose Boosted Mean Shift Clustering (BMSC) to solve these problems.[28]
Adding to the previously described advantages of DBSCAN, BMSC comes to override the limitations of Mean Shift and DBSCAN, while retaining their nonparametric nature. We seek to capture the underlying group structure of the data by selecting a subset of the data that provides the skeleton of the clusters.
In order to explain how BMSC works, we start by detailing the Mean Shift.

Mean Shift [10]:

The Mean Shift algorithm is a non-parametric clustering technique that estimates the number of clusters directly from the data, and is able to find irregularly shaped clusters.
It assigns data points to groups iteratively by moving the points to the mode. The mode is the point representing the center of density in the selected region. This mode point is

detected by performing an estimate of the kernel density of the mixture in question as shown in the attached figure 4.3.
Points that converge to the same mode are considered members of the same group.
The key parameter of the Mean Shift is then the kernel bandwidth. Its value can affect the performance of the mean shift and is difficult to define.

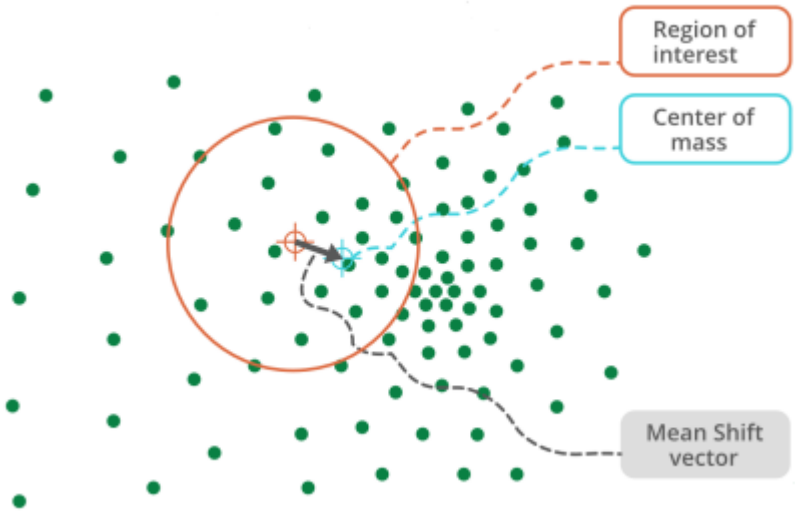

Figure 4.3: Mean Shift [URL9]

As shown in Figure 4.3, the steps for applying Mean Shift are in the order :

1. The selection of an arbitrary region (the region in red on the figure).
2. The calculation of the center of density of the region (the blue point on the figure) and the generation of the mean shift vector.
3. The displacement of the center of the region along the direction of the mean shift vector until coincidence.
4. Repeat steps 2 and 3 until convergence.

4.1.2.1 Hyperparameters of BMSC

The BMSC hyperparameters are the combination of Mean Shift and DBSCAN hyperparameters :

- kernel bandwidth
- Eps
- MinPts

4.1.2.2 Application steps

The application steps of the BMSC algorithm can be illustrated through the pseudo-code in figure 4.4:

Algorithm 1. Boosted Mean Shift Clustering

Input: $\mathcal{X}$, *width*, *height*, *h*, *Eps*.
Output: The final clustering result cl_final.

```
1: InitializeGrid(X, width, height); //Distribute X over I = width × height cells.
2: iModes ← ∅;                       //Initialize the set of intermediate modes.
3: counter ← 1;
4: repeat
5:    for j ← 1 to I do
6:       newiModes ← MeanShift(cellData_j,h);
7:       iModes.append(newiModes);
8:    end for
9:    ConfidenceAssignment();  // Assign confidence values to points in each cell
      via Eqs. (5) and (6).
10:   for j ← 1 to I do
11:      CollectedData ← CollectNeighborData(j) ∪ cellData_j;
12:      cellData_j ← WeightedSampling(CollectedData);   // Update cellData_j.
13:   end for
14:   [cl_iModes, numberOfClustersDetected] ← DBSCAN(iModes,Eps);
      //cl_iModes is the clustering result of iModes.
15:   if (numberOfClustersDetected==lastnumberOfClustersDetected) then
16:      counter++;
17:   else
18:      counter ← 1;
19:   end if
20: until counter == 3
21: cl_final ← DataAssignment(X, iModes,cl_iModes);      //Assign points in X.
22: return cl_final.
```

Figure 4.4: BMSC algorithm [28]

In this algorithm, we start by distributing the data evenly across the cells of a grid that the user defines. In our project, we have defined a 3*3 grid.
Then, in a first step we apply the Mean Shift algorithm on the data of each cell of the grid and consequently we obtain a list of mode points called the intermediate modes (iModes). after that, we redistribute the data of each cell according to the following mechanism :

Each cell of the grid interacts with a finite number of cells in its surroundings, which form its neighborhood according to a previously defined neighborhood structure. The different neighborhood structures are illustrated in figure 4.5:

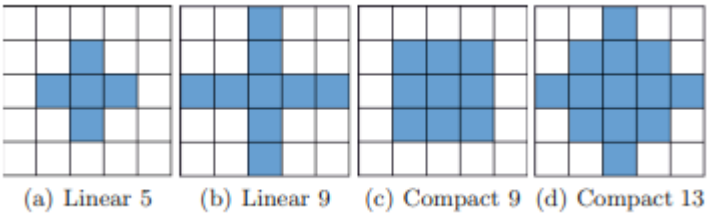

Figure 4.5: Neighborhood structures [28]

In our project, given the size of our grid, we use the linear(5) neighborhood structure.

Afterwards, in a second step of the re-sampling mechanism, we compute the distance that separates all the points of the parent cell and its neighborhood from the iModes detected by the Means shift. We assign the points to the cell depending on the distance that separates them from the iModes of the cell while keeping the same initial cell size.
At this stage, we assign each point to the iMode that is closest to it.
Then we apply DBSCAN on the list of iModes in order to collect the iModes that are densely packed and subsequently at a lower level we build clusters of the original data points. We repeat these same steps until DBSCAN detects the same number of clusters for three consecutive iterations.
The benefits of BMSC are :

- BMSC is recognized by better test results than DBSCAN. This will be proven in the results that will be presented in the next last chapter.
- The execution time is also acceptable compared to the computation time taken by DBSCAN alone because the latter will only be applied on a reduced number of data points and not on the whole database.
- The authors of BMSC proved in their paper that compared to Means Shift and DBSCAN separately, BMSC is not as sensitive to the choice of hyper-parameters and this by varying the hyper-parameters one by one and by controlling the evolution of the results of the algorithm.

4.2 Modeling the codependent call approach:

In this second approach, we apply the two algorithms: DBSCAN in a first part and BMSC in a second part.
We will conduct a comparative study of the results in the next chapter of our report.

4.3 Modeling of the graph theory approach

This third and last approach models the problem using graph theory. Starting from the weighted directed graph illustrated in figure 4.6 where the red nodes represent the classes of the application and the edges represent a relationship according to the class similarity metric detailed in the third chapter of the report.
To solve our problem, we apply two community detection algorithms that we present in the following parts.

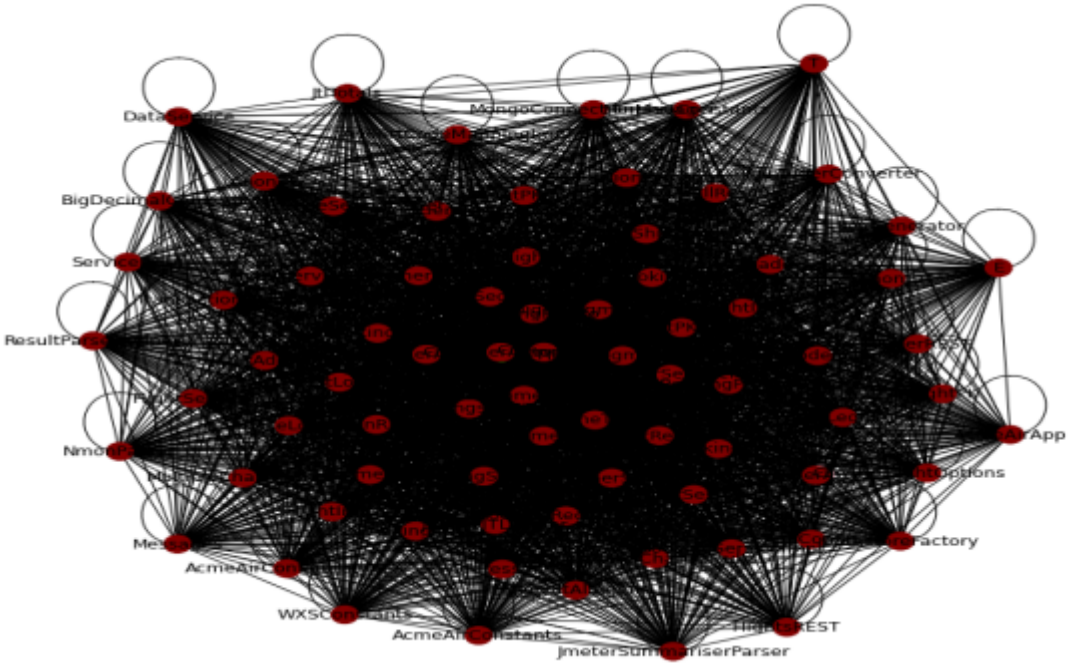

Figure 4.6: Weighted graph showing the relationships between the classes of the Acmeair application

4.3.1 Girvan-Newman

The Girvan-Newman is an algorithm for detecting and analyzing community structure by splitting relies on the iterative elimination of edges that have the largest number of shortest paths between the nodes that cross them. By eliminating the edges of the graph one by one, the network decomposes into smaller pieces, called communities.[44]
It can be summarized in the following steps :

- For each edge of a graph, we compute the interdependence centrality of the edge. Edge interdependence centrality is a measure of the centrality of an edge in a network based on the number of shortest paths that pass through the given edge. It thus identifies edges in the network that are crucial for in- formation flows.

In our case, these values are those of the class similarity that we have calculated from the fact that classes that are similar to each other will have a higher class similarity value is therefore an important flow of information that passes through the edge that connects them.

- We remove the edge with the smallest interdependence centrality.
- We repeat the two previous steps until there are no more edges.

4.3.2 Louvain

Louvain is a community detection algorithm that is part of the agglomerative approaches. It is a heuristic method based on the optimization of modularity. The algorithm works in 2 steps [8] :

- Assign each node to be in its own community.
- Try to find the maximum positive modularity gain by moving each node to all neighboring communities. If no positive gain is obtained, the node remains in its original community.

Conclusion

In this chapter, we started by presenting the modeling phase which concerns each of the developed resolution approaches by detailing the used algorithms.

Chapter 5

Implementation and results

Introduction

In this last chapter, we present in a first section the realization environment of our solution. Then, we show our results by comparing them to those distilled from different approaches in the literature and we end with a discussion highlighting our contributions during this project.

5.1 Work environment

In this section we present the hardware and software environment necessary to achieve our goal.

5.1.1 Software environment

In this section, we present the different technologies used to implement this project.

- **Python:** It is an object-oriented, interpreted, multi-paradigm and multi-platform programming language. It is simple to master and can be used in a wide variety of fields, including web development, data science, AI, security, etc. Therefore, it is one of the most widely used languages[URL15].

- **Scikit-learn (Sklearn) :** This is the richest and most useful library for machine learning in Python. It offers a range of powerful tools f o r machine learning and statistical modeling, as well as classification, regression, clustering and dimensional reduction, through a consistent interface in Python.[URL13].

- **Natural Language ToolKit (NLTK) :** NLTK is a leading platform for creating Python programs for working with data built in from human language. It provides

easy-to-use interfaces and a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, syntactic analysis and semantic reasoning. [URL10].

- **Understand :** It is a static analysis tool of the source code of applications allowing to generate a set of information for our case we used it to generate the call matrices between the classes of the application.[URL14].

5.1.2 Hardware environment

In order to implement our solution, we used different resources :

- **Google Colaboratory :** It is a platform of free resources offered by Google. It has : 1 GPU (Tesla T4) With 15GB [URL5].

- **ASUS :** For the development of the project, we used an ASUS personal computer with these features :
Processor : Intel i7-7500U CPU 2.70GHz | Operating System : Windows 10. Mémory : 8 GB RAM | Hard Disk : 1 To.

5.2 Experimentation

In order to test the developed approaches and draw results from them, we must, at this stage, fix the hyper-parameters of each approach.
We used the same hyper-parameters for the application of the first two approaches :

- **MinPts:** Let be the hyper-parameter corresponding to the DBSCAN algorithm. In order to optimize our solution and to better satisfy the evaluation metrics, this hyper-parameter has been fixed during all the experimentation to **MinPts=5**.
Indeed, the NED evaluation criterion defined in Chapter 2 indicates that a micro service is not considered very small if it contains at least five classes.

- **Eps:** The limiting distance between two classes for one to be considered in the neighborhood of the otheris the most important DBSCAN parameter to choose appropriately for our data set.
We varied the Eps values from 0 to 1 with a step equal to 0.05 to obtain the best results which we present in the next section for the value of **Eps = 0.65**.

- **Kernel bandwidth :** This hyper-parameter was determined using the $estimated_bandwidth$ function which uses a heuristic based on the median of all pairwise distances to generate the best one that matches our input data.

On the other hand, the hyper-parameters of the third and last approach have been determined from the literature. Indeed, Sellami and al.[42] in their articles have conducted a comparative study of the variation of the parameters in order to derive the best value. Therefore and based on their study, we set these hyper-parameters as follows :

- **Alpha α :** Being the coefficient that represents the weight of the structural analysis in the class similarity metric, hence in determining the relationship between each pair of classes.
  According to Sellami and al.[42] the optimal values for the hyper-parameter α are in the interval [0,45, 0,55].Therefore we have fixed $\alpha = \mathbf{0.5}$.
  This indicates that semantic similarity and structural similarity are equally important in determining the valuation of the relationship between two classes.

- **Beta β :** Therefore, by satisfying the condition $\alpha + \beta = 1$ β we obtain **0.5**.

The algorithms that we have applied for the detection of communities within the graph only require as parameters the graph and the list of weights of the edges.
After fixing the hyper-parameters of each of the developed approaches, we move to our third and final research question :

Research Question 3 : How does our solution compare to the different approaches extracted from the literature that are being solved for our microservices extraction problem?

5.3 Results

In this last section, we present the results of our solution, which corresponds to the fifth phase of the CRISP-DM methodology entitled "Evaluation". We conclude this section with a discussion comparing our final results with those of synthesized approaches in the literature.

To begin with, we summarize in the two attached tables 5.1 and 5.2 the different results of the state-of-the-art approaches applied respectively to the Acmeair and Daytrader applications.
The first column of the table contains the evaluation metrics that we presented in the second chapter and the rest of the columns present the results of different approaches in the literature.

Following the detected values for the different approaches presented in Table 5.1, we can notice that the CoGCN approach determined in the paper "HierDecomp" [42] has the value NED=1 which is an extreme result. This proves that all microservices formed by this approach are very small or very large. Therefore, we decide to eliminate it from

Acmeair	Mono2Micro-IBM	Bunch-HierDecomp	GNN	FoSCI-HierDecomp	MEM-HierDecomp
SM	0.28	0.25	0.42	**0.29**	0.28
IFN	1.3	1.5	0.8	1.7	**0.8**
NED	0.7	**0.3**	1	0.7	0.35
ICP	0.5	0.5	0.62	0.72	**0.1**

Table 5.1: Summary table of evaluation results of state of the art approaches for the Acmeair application

the comparison approaches because it does not respect one of the main objectives of the migration. This famous problem is known in literature under the name of the "Problem of grains and borders"

DayTrader	Mono2Micro-IBM	Bunch-HierDecomp	GNN	FoSCI-HierDecomp	MEM-HierDecomp
SM	**0.53**	0.15	0.5	0.29	0.3
IFN	3	**0.1**	13	6	3.5
NED	**0.3**	0.65	0.65	0.58	1
ICP	**0.1**	0.5	0.3	0.8	0.25

Table 5.2: Summary table of the results of the evaluation of the state of the art approaches for the DayTrader application

For the same reasons cited above, we eliminate the MEM approach from table 5.2.

5.3.1 Results of the naive approach and the codependent calls approach

The following table summarizes the results obtained for our first two approaches using both DBSCAN and BMSC algorithms for each.

As we can notice from Table 5.3 of the Acmeair application, the best result for this application was generated by the second approach of codependent calls.
On the other hand, for this same approach of codependent calls, BMSC has shown to be more efficient than DBSCAN and this by comparing the values obtained for the different evaluation metrics. Indeed, the best result for SM is 0.44 resulting from the application of BMSC for the second resolution approach. The same is true for all the other metrics.

Since these evaluations, we find that the second approach is more efficient than the naive one, due to the refinement of the class representation criterion as explained in the third chapter of our report.

Acmeair	Algorithms	Naive approach	Co-dependent call approach
SM	DBSCAN	0.16	0.17
	BMSC	0.42	**0.44**
IFN	DBSCAN	0.5	1
	BMSC	4.8	**0.2**
NED	DBSCAN	0.8	1
	BMSC	0.8	**0.8**
ICP	DBSCAN	0.48	0.56
	BMSC	0.58	**0.4**

Table 5.3: Summary table of the results of the first two approaches - Acmeair

A further evaluation shows that BMSC performs better than DBSCAN when the data have a wide range of density. This confirms the results shown by Ren et al [28].
In a second phase, we compare our results with those of the literature presented in table 5.1.
Our solution proves the best result for SM and NED metrics with 0.44 and 0.2 respectively compared to the best values found which are SM=0.42 for CoGCN and IFN=0.8 for this same approach.

DayTrader	Algorithms	Naive approach	Co-dependent calls approach
SM	DBSCAN	0.42	0.2
	BMSC	**0.7**	**0.7**
IFN	DBSCAN	0	2.2
	BMSC	11	**1.6**
NED	DBSCAN	1	0.8
	BMSC	0.85	**0.8**
ICP	DBSCAN	0.57	0.52
	BMSC	0.54	**0.5**

Table 5.4: Summary table of the results of the first two approaches - DayTrader

DayTrader also confirms through the results generated the performance of the second approach for the application of the BMSC algorithm compared to the first naive and DBSCAN.
As shown in Table 5.4 of the DayTrader application, we obtained the same best SM value for 0.7 when applying BMSC for both approaches. This puts both approaches on the same

line of efficiency but if we move forward in our evaluation by exploiting the other aspects of the newly trained system, we notice that the preferred results are those of the second approach with NFI = 1.6 against NFI=11 during the first approach. Similarly, NED having its best value for 0.8 and finally ICP for the value of 0.5.

Also, DayTrader achieves the best results for the SM, IFN metrics compared to all approaches except the Bunch method.

5.3.2 Results of the graph approach

The following table summarizes the results obtained for our last reso- lution approach based on graph theory.

Metrics	Algorithms	**Acmeair**	**DayTrader**
SM	Girvan Newman	0.7	0.3
	Louvain	0.5	0.4
IFN	Girvan Newman	1	0.6
	Louvain	1.4	0.8
NED	Girvan Newman	1	0.8
	Louvain	0.3	0.6
ICP	Girvan Newman	0.1	0.52
	Louvain	0.6	0.85

Table 5.5: Summary table of the results of the graph approach

From table 5.5, we notice that for the Acmeair application, the Girvan-Newman algorithm generates the best results. The value of NED being extreme equal to 1 pushes us to further detail the evaluation and this through figure 5.1.

Note that the nodes with the same color in the figure belong to the same cluster, in other words the classes sharing the same color of nodes belong to the same microservice. The figure shows that the Girvan-Newman algorithm and to maximize the value of structural modularity tried to gather all the classes in the same microservice except two that were each assigned to a microservice. This resulted in the formation of one very large microservice and two very small microservices which does not meet the NED metric.
On the other hand and for the same Acmeair application, the Louvain community detection algorithm generated four medium-sized clusters as shown in Figure 5.2 thus justifies the value obtained for the NED metric which is equal to 0.4.

A comparative study between the results of approach 2 and approach 3 proves that the codependent calls approach tends to generate the best results in terms of structural modularity but the graph approach focuses much more on minimizing the metrics considering even more the coupling.

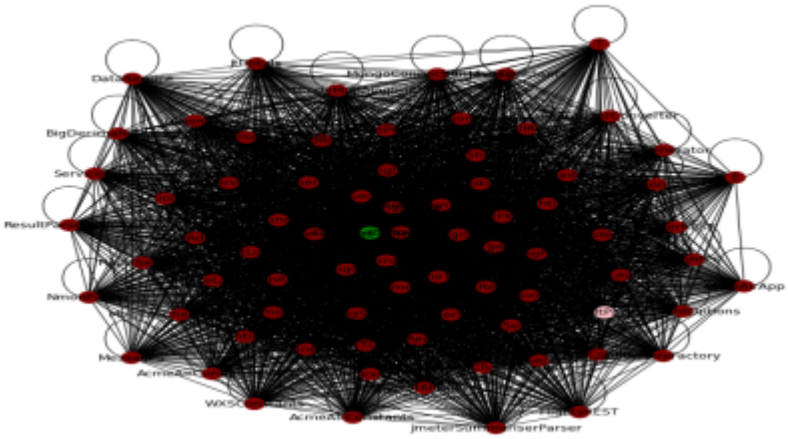

Figure 5.1: Community graph of the application Acmeair using Girvan Newman algorithm

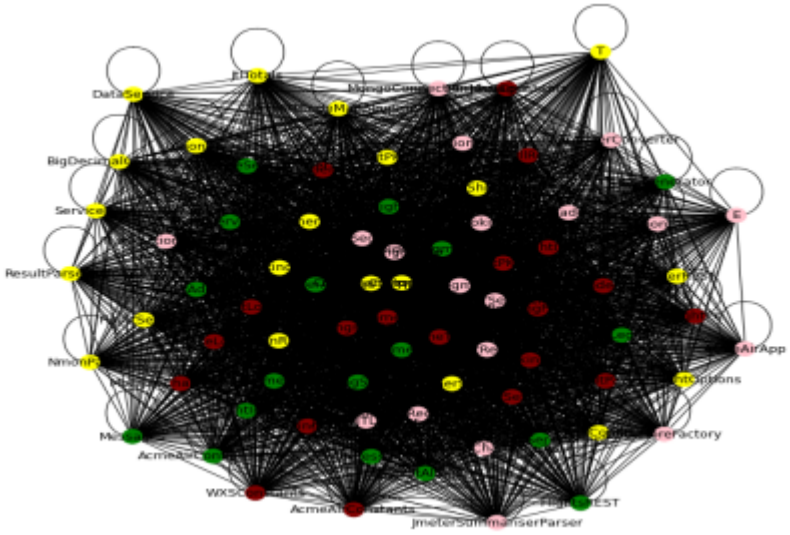

Figure 5.2: Community graph of the application Acmeair using Louvain algorithm

5.3.3 Discussion

Comparing to the reference approaches, our solution dealing with the problem from a clustering point of view using the BMSC algorithm outperforms all other solutions in the literature in terms of structural modularity for both Acmeair and DayTrader test applications. Nevertheless, it is worse than some other methods for the rest of the metrics.

This problem of unsatisfiability of all evaluation metrics is manifested in all approaches as we note from Tables 5.1 and 5.2.
No approach in the literature presents the best performances for all the proposed evaluation metrics but to be able to conclude about the reliability of the approaches and according to the literature an approach is classified as acceptable if it reaches good results for the structural modularity provided that it does not contain extreme values for NED and IFN. Therefore, we can conclude that our second approach based on the consideration of code-dependent calls by applying the BMSC algorithm satisfies the resolution of the migration problem by presenting performances that exceed those of the literature approaches presented in Tables 5.1 and 5.2.

Conclusion

During this last chapter of our report, we started by presenting our working environment with these two software and hardware parts. Then, we summarized the results of different approaches in the literature and presented our own in a second part. Finally, we concluded the chapter with a discussion of the results.

Conclusion and perspectives

In this report, we present an approach to decompose monolithic applications into microservices based on a static analysis of the source code of the latter while treating the problem as a clustering problem.

We began by introducing the general context of the project to lay the groundwork. Then, we have introduced the different key concepts necessary for a good understanding of the theme. We also reviewed existing approaches using a well-defined research methodology. This review was the key to establishing our approach. It is conducted to inquire about the main results of existing similar works and their limitations, which helped us to define the contributions of the work and to make judicious choices on the methodology of treatment of the problem. Finally, the report details the work done starting with the understanding, collection and processing of the data up to the modeling part. We conclude our report with the results of our approach and a comparison with existing approaches.

At the end of this course, we can conclude that we have obtained good results since we have outperformed existing solutions in many evaluation metrics. The best results we have detected are those generated by the second approach using codependent calls between classes of the application precisely by applying the BMSC clustering algorithm.

During this project, we encountered many problems when establishing the decomposition approach such as the definition of the decomposition criteria and the level of granularity of the solution as well as the method and metrics to be used for the evaluation of the results, from which the need for the research was born.

Therefore, we could continue this project by: further developing a complement to the evaluation strategy of our project that corresponds to a semi-automatic decomposition process. This process will be considered as the basis for per-application comparison since the evaluation metrics used in the literature do not cover all aspects of the monolithic application decomposition.
In addition, we can continue to exploit graph theory by testing the notion of deep community detection.
Moreover, the choice of the granularity level is a decisive step of the project, so we can

push the solution even further to not consider classes as a basis for decomposition but rather methods or functions of the monolith.
Also, we can make our solution hybrid by injecting a dynamic analysis of the source code based on the fact that the static analysis does not provide all the elements necessary for the good understanding of the functionalities and their interactions during the execution of the application.
A final perspective is to not only recommend a way to decompose the monolith but also to recommend a way to rewrite the application code if necessary.

Bibliography

[1] Nuri Almarimi, Ali Ouni, Salah Bouktif, Mohamed Wiem Mkaouer, Raula Gaikovina Kula, and Mohamed Aymen Saied. Web service api recommendation for automated mashup creation using multi-objective evolutionary search. *Applied Soft Computing*, 85:105830, 2019.

[2] Omar Benomar, Hani Abdeen, Houari Sahraoui, Pierre Poulin, and Mohamed Aymen Saied. Detection of software evolution phases based on development activities. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 15–24. IEEE, 2015.

[3] 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. ACM, 2021.

[4] Hugo Brunelière, Jordi Cabot, Grégoire Dupé, and Frédéric Madiot. MoDisco: A model driven reverse engineering framework. *Information and Software Technology*, pages 1012–1032, 2014.

[5] Ricardo J. G. B. Campello, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *WIREs Data Mining Knowl Discov*, 10:e1343, 2020.

[6] Francisco Handrick da Costa, Ismael Medeiros, Thales Menezes, João Victor da Silva, Ingrid Lorraine da Silva, Rodrigo Bonifácio, Krishna Narasimhan, and Márcio Ribeiro. Exploring the use of static and dynamic analysis to improve the performance of the mining sandbox approach for android malware identification.

[7] Lorenzo De Lauretis. From monolithic architecture to microservices architecture. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 93–96, 2019.

[8] Pasquale De Meo, Emilio Ferrara, Giacomo Fiumara, and Alessandro Provetti. Generalized louvain method for community detection in large networks. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 88–93, 2011.

[9] MohammadHadi Dehghani, Shekoufeh Kolahdouz-Rahimi, Massimo Tisi, and Dalila Tamzalit. Facilitating the migration to the microservice architecture via model-driven reverse engineering and reinforcement learning. *Softw Syst Model*, 21:1115–1133, 2022.

[10] Konstantinos G. Derpanis. Mean shift clustering.

[11] Utkarsh Desai, Sambaran Bandyopadhyay, and Srikanth Tamilselvam. Graph neural network to dilute outliers for refactoring monolith application.

[12] Jens Dietrich, Catherine McCartin, Ewan Tempero, and Syed M Ali Shah. On the existence of high-impact refactoring opportunities in programs. *Computer Science*, 122, 2012.

[13] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486:75–174, 2010.

[14] Martin Fowler, Kent Beck, and John Brant. Refactoring - improving the design of existing code. *Addison-Wesley*, page 337, 2019.

[15] Jonas Fritzsch, Justus Bogner, Alfred Zimmermann, and Stefan Wagner. From monolith to microservices: A classification of refactoring approaches. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, volume 11350, pages 128–141. Springer International Publishing, 2019.

[16] Maxime Gallais-Jimenez, Hoan A Nguyen, Mohamed Aymen Saied, Tien N Nguyen, and Houari Sahraoui. Api misuse detection an immune system inspired approach. *arXiv preprint arXiv:2012.14078*, 2020.

[17] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. Service cutter: A systematic approach to service decomposition. In Marco Aiello, Einar Broch Johnsen, Schahram Dustdar, and Ilche Georgievski, editors, *Service-Oriented and Cloud Computing*, volume 9846, pages 185–200. Springer International Publishing, 2016.

[18] Kendall K. Hall, Sarah Shoemaker-Hunt, Lynn Hoffman, Sonja Richard, Elizabeth Gall, Elizabeth Schoyer, Dana Costar, Bryan Gale, Gordon Schiff, Kristen Miller, Tara Earl, Nicole Katapodis, Cori Sheedy, Brandy Wyant, Olivia Bacon, Andrea Hassol, Stephanie Schneiderman, Meghan Woo, Lisa LeRoy, Eleanor Fitall, Anna Long, Aline Holmes, Jennifer Riggs, and Andrea Lim. *PRISMA Flow Diagrams*. Agency for Healthcare Research and Quality (US), 2020-03.

[19] Samuel Huppe, Mohamed Aymen Saied, and Houari Sahraoui. Mining complex temporal api usage patterns: an evolutionary approach. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 274–276. IEEE, 2017.

[20] Anup; K. Kalia, Xiao Jin, Krishna Rahul, Sinha Saurabh, Vukovic Maja, and Banerjee Debasish. 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*, 2021.

[21] Hatice Koç, Ali Mert Erdoğan, Yousef Barjakly, and Serhat Peker. UML diagrams in software engineering research: A systematic literature review. In *The 7th International Management Information Systems Conference*. MDPI, 2021.

[22] Cai-zhi Liu, Yan-xiu Sheng, Zhi-qiang Wei, and Yong-Quan Yang. Research of text classification based on improved TF-IDF algorithm. In *2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*, 2018.

[23] Vijayarani Mohan. Preprocessing techniques for text mining - an overview. *International Journal of Computer Science & Communication Networks*, 2015.

[24] Mohamed Mouine and Mohamed Aymen Saied. Event-driven approach for monitoring and orchestration of cloud and edge-enabled iot systems. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 273–282. IEEE, 2022.

[25] Suhaib Mujahid, Diego Elias Costa, Rabe Abdalkareem, Emad Shihab, Mohamed Aymen Saied, and Bram Adams. Toward using package centrality trend to identify packages in decline. *IEEE Transactions on Engineering Management*, 2021.

[26] Sam Newman. Monolith to microservices. *O'Reilly*, page 273, 2019.

[27] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and David Moher. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 2021.

[28] Yazhou Ren, Uday Kamath, Carlotta Domeniconi, and Guoji Zhang. Boosted mean shift clustering. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8725, pages 646–661. Springer Berlin Heidelberg, 2014.

[29] Islem Saidani, Ali Ouni, Mohamed Wiem Mkaouer, and Aymen Saied. Towards automated microservices extraction using muti-objective evolutionary search. In *International Conference on Service-Oriented Computing*, pages 58–63. Springer, 2019.

[30] Mohamed Aymen Saied, Hani Abdeen, Omar Benomar, and Houari Sahraoui. Could we infer unordered api usage patterns only using the library source code? In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 71–81. IEEE, 2015.

[31] Mohamed Aymen Saied, Omar Benomar, Hani Abdeen, and Houari Sahraoui. Mining multi-level api usage patterns. In *2015 IEEE 22nd international conference on software analysis, evolution, and reengineering (SANER)*, pages 23–32. IEEE, 2015.

[32] Mohamed Aymen Saied, Omar Benomar, Hani Abdeen, and Houari Sahraoui. Mining multi-level API usage patterns. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2015.

[33] Mohamed Aymen Saied, Omar Benomar, and Houari Sahraoui. Visualization based api usage patterns refining. In *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, pages 155–159. IEEE, 2015.

[34] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. Automated inference of software library usage patterns. *arXiv preprint arXiv:1612.01626*, 2016.

[35] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software*, 145:164–179, 2018.

[36] Mohamed Aymen Saied, Erick Raelijohn, Edouard Batot, Michalis Famelis, and Houari Sahraoui. Towards assisting developers in api usage by automated recovery of complex temporal patterns. *Information and Software Technology*, 119:106213, 2020.

[37] Mohamed Aymen Saied and Houari Sahraoui. A cooperative approach for combining client-based and library-based api usage pattern mining. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10. IEEE, 2016.

[38] Mohamed Aymen Saied, Houari Sahraoui, Edouard Batot, Michalis Famelis, and Pierre-Olivier Talbot. Towards the automated recovery of complex temporal api-usage patterns. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1435–1442, 2018.

[39] Mohamed Aymen Saied, Houari Sahraoui, and Bruno Dufour. An observational study on api usage constraints and their documentation. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 33–42. IEEE, 2015.

[40] Khaled Sellami, Ali Ouni, Mohamed Aymen Saied, Salah Bouktif, and Mohamed Wiem Mkaouer. Improving microservices extraction using evolutionary search. *Information and Software Technology*, page 106996, 2022.

[41] Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni. A hierarchical dbscan method for extracting microservices from monolithic applications. In *The International Conference on Evaluation and Assessment in Software Engineering 2022*, pages 201–210, 2022.

[42] Khaled Sellami, Mohamed Aymen Saied, and Ali Ouni. A hierarchical DBSCAN method for extracting microservices from monolithic applications. In *The International Conference on Evaluation and Assessment in Software Engineering 2022*, pages 201–210. ACM, 2022.

[43] Khaled Sellami, Mohamed Aymen Saied, Ali Ouni, and Rabe Abdalkareem. Combining static and dynamic analysis to decompose monolithic application into microservices. In *International Conference on Service-Oriented Computing*, pages 203–218. Springer, 2022.

[44] Seunghyeon Moon, Jae-Gil Lee, and Minseo Kang. Scalable community detection from networks by computing edge betweenness on MapReduce. In *2014 International Conference on Big Data and Smart Computing (BIGCOMP)*, pages 145–148. IEEE, 2014.

[45] Anas Shatnawi, Hudhaifa Shatnawi, Mohamed Aymen Saied, Zakarea Al Shara, Houari Sahraoui, and Abdelhak Seriai. Identifying software components from object-oriented apis based on dynamic analysis. In *Proceedings of the 26th Conference on Program Comprehension*, pages 189–199, 2018.

[46] Harsh Vardhan Singh, Ashwin Girdhar, and Sonika Dahiya. A literature survey based on DBSCAN algorithms. In *2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 751–758, 2022.

[47] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. Deploying microservice based applications with kubernetes: Experiments and lessons learned. In *2018 IEEE 11th international conference on cloud computing (CLOUD)*, pages 970–973. IEEE, 2018.

[48] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. Kubernetes as an availability manager for microservice applications. *arXiv preprint arXiv:1901.04946*, 2019.

[49] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. Microservice based architecture: Towards high-availability for stateful applications

with kubernetes. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, pages 176–185. IEEE, 2019.

[50] Leila Abdollahi Vayghan, Mohamed Aymen Saied, Maria Toeroe, and Ferhat Khendek. A kubernetes controller for managing the availability of elastic microservice based stateful applications. *Journal of Systems and Software*, 175:110924, 2021.

[51] R. Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, page 11, 01 2000.

[52] J. Wu, A.E. Hassan, and R.C. Holt. Comparison of clustering algorithms in the context of software evolution. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*, pages 525–535, 2005.

[53] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16:645–678, 2005.

Netography

[URL1] Acme air sample and benchmark. https://github.com/acmeair/acmeair, visité le 2022-08-11.

[URL2] Connected papers | find and explore academic papers. https://www.connectedpapers.com/, visité le 2022-08-23.

[URL3] Définition | programmation orientée objet - POO - programmation par objet | futura tech. https://www.futura-sciences.com/tech/definitions/informatique-programmation-orientee-objet-19301/, visité le 2022-08-20.

[URL4] Getting started with community detection in graphs and networks. https://www.analyticsvidhya.com/blog/2020/04/community-detection-graphs-networks/, visité le 2022-08-12.

[URL5] Google colab. https://research.google.com/colaboratory/faq.html, visité le 2022-08-31.

[URL6] Google scholar. https://scholar.google.com/, visité le 2022-08-26.

[URL7] L'ingénieur logiciel. https://www.polymtl.ca/gigl/la-profession-ding%C3%A9nieur-en-g%C3%A9nie-informatique-ou-g%C3%A9nie-logiciel/aper%C3%A7u-de-la-profession/lingenieur-logiciel, visité le 2022-08-18.

[URL8] Microservices. https://martinfowler.com/articles/microservices.html, visité le 2022-08-18.

[URL9] ML | mean-shift clustering. https://www.geeksforgeeks.org/ml-mean-shift-clustering/, visité le 2022-08-22.

[URL10] NLTK :: Natural language toolkit. https://www.nltk.org/, visité le 2022-08-31.

[URL11] Qu'est-ce que le machine learning ? https://www.oracle.com/ch-fr/artificial-intelligence/machine-learning/what-is-machine-learning/, visité le 2022-08-19.

[URL12] sample.daytrader7 tech. `https://github.com/WASdev/sample.daytrader7`, visité le 2022-08-19.

[URL13] scikit-learn: machine learning in python — scikit-learn 1.1.2 documentation. `https://scikit-learn.org/stable/`, visité le 2022-08-31.

[URL14] Understand: An IDE and static code analysis tool by SciTools. `https://www.scitools.com`, visité le 2022-08-23.

[URL15] Welcome to python.org. `https://www.python.org/`, visité le 2022-08-31.

[URL16] Atlassian. Microservices et architecture monolithique. `https://www.atlassian.com/fr/microservices/microservices-architecture/microservices-vs-monolith`, visité le 2022-08-19.