A MICROSERVICE-BASED ARCHITECTURE FOR AN ONLINE PRODUCT REVIEW ANALYSIS SYSTEM

MEHDI MOHAMMADI

FACULTY OF COMPUTER SCIENCE AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2021

A MICROSERVICE-BASED ARCHITECTURE FOR AN ONLINE PRODUCT REVIEW ANALYSIS SYSTEM

MEHDI MOHAMMADI

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SOFTWARE ENGINEERING

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2021

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **Mehdi Mohammadi**

Matric No: **WGC150028**

Name of Degree: **Master of Software Engineering**

Title of Dissertation ("this Work"): **A Microservice-Based Architecture for an Online Product Review Analysis System**

Field of Study: **Software Architecture**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date: 2021-01-24

Subscribed and solemnly declared before,

Witness's Signature Date: 6-2-2021

Name:

Designation:

A MICROSERVICE-BASED ARCHITECTURE FOR AN ONLINE PRODUCT REVIEW ANALYSIS SYSTEM

ABSTRACT

Since the emergence of Web 2.0 in late 1999, online product reviews have played a significant role in the e-commerce channels, social media, online forums, and online communities. The majority of studies in the area of product review analysis (PRA) systems have focused on evaluating and improving the techniques for review analysis to increase efficiency, accuracy, effectiveness, usefulness, ranking, spam detection, and feature extraction. The architecture of PRA systems is not the primary concern of research in this field. In this study, some of the implicit and explicit architectures of scientifically reported PRA systems were extracted to examine how the architectures have been evolving. One of the recently employed architectures in this area is the microservice architecture, aimed at achieving high flexibility and maintainability by developing loosely-coupled microservices. Current PRA systems do not fully utilise microservice architecture to achieve low-coupling in design. This research focuses on evaluating existing PRA architectures and proposing improvements on a microservice-based PRA architecture (Viscovery) in terms of reducing coupling. To evaluate the proposed architecture, two different methods, one of them is a quality model (MM4S model) and the other one involving descriptive techniques, were adopted. A prototype PRA system was developed based on the proposed architecture. The prototype system can retrieve reviews from multiple e-commerce websites and analyse the review text. The evaluation shows that the prototype system achieves lower coupling, as compared to a benchmarked microservice-based PRA system.

Keywords: Product review analysis systems, sentiment analysis, architecture, microservice

SENI BINA BERASASKAN PERKHIDMATAN MIKRO UNTUK SUATU SISTEM ANALISIS ULASAN PRODUK DALAM TALIAN

ABSTRAK

Sejak kemunculan Web 2.0 pada akhir tahun 1999, ulasan produk dalam talian telah memainkan peranan yang sangat penting dalam saluran e-dagang, media sosial, forum dalam talian dan komuniti dalam talian. Kebanyakan penyelidikan dalam bidang sistem analisis ulasan produk (PRA) telah menumpukan perhatian untuk menilai dan memperbaiki teknik analisis termasuk meningkatkan kecekapan, ketepatan, keberkesanan, kebergunaan, menentukan kedudukan, pengesanan spam dan pengekstrakan ciri. Walau bagaimanapun, seni bina untuk sistem-sistem ini bukan pertimbangan utama dalam bidang penyelidikan tersebut. Kajian ini mengekstrak seni bina sistem PRA yang tersirat dan tersurat yang dilaporkan secara saintifik untuk mengkaji bagaimana seni bina berkenaan telah berkembang. Salah satu seni bina terkini yang digunakan adalah seni bina perkhidmatan mikro, yang bertujuan untuk mencapai fleksibiliti dan penyelenggaraan yang tinggi dengan membangunkan perkhidmatan mikro yang digandingkan secara longgar. Sistem-sistem PRA semasa tidak menggunakan sepenuhnya seni bina perkhidmatan mikro untuk menghasilkan sistem bergandingan rendah. Kajian ini memfokus pada penilaian seni bina PRA sedia ada dan mencadangkan penambahbaikan bagi satu seni bina berasaskan perkhidmatan mikro (Viscovery) dari segi penurunan gandingan. Untuk menilai seni bina yang dicadangkan, dua kaedah berbeza, yang salah satunya bersifat kuantitatif (model MM4S) dan yang satu lagi melibatkan teknik deskriptif dan visualisasi telah digunakan. Satu sistem prototaip PRA dibangunkan berdasarkan seni bina yang dicadangkan. Sistem prototaip ini dapat memperoleh dan menganalisis ulasan dari pelbagai laman web e-dagang. Penilaian menunjukkan pencapaian gandingan yang lebih rendah dalam sistem prototaip yang

dibangunkan berbanding dengan sebuah sistem berasaskan perkhidmatan mikro yang ditandaraskan.

Kata kunci: Sistem analisis ulasan produk, analisis sentiment, seni bina, perkhidmatan mikro

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Associate Professor Dr. Chiew Thiam Kian, who supported me throughout this dissertation on the journey. All of the hints, guidance, and direction that I got from him were most useful. I appreciate my wife Maryam and my daughter Avina for supporting me even though I spent most of our family time on this study. I appreciate my parents' courage and support, especially my father who told me, "*Never stop learning"*.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

CCM	:	Conceptual coupling between methods
CoCoME	:	Common component modelling example
DDD		Domain-driven development
EWOM	:	Electronic word of mouth
PRA	:	Product review analysis
POS	:	Part of speech
QA	:	Quality assurance
SDLC	:	Software development life cycle
SOA	:	Service-oriented architecture
UGC	:	User-generated content

CHAPTER 1: INTRODUCTION

1.1 Background

Product reviews are a type of user-generated content (UGC) obtained through online e-commerce channels and community forums. UGCs may describe a consumer's purchase experience, the product quality description, rating, quality, service, and delivery. The reviews are significant for assessing customer satisfaction and making any judgments about products, especially for new customers. Product reviews' areas of application include anticipating a customer's purchase intentions, readership, sales revenue and decision-making processes, marketing strategies, as well as improving product or service quality and merchant sales.

As a business grows, the volume of product reviews also increases, making it harder for customers to read all the reviews due to information overloading. On the other hand, obtaining these reviews in a timely manner and analysing them is significant for all the stakeholders involved, including the business owners, vendors, and suppliers, as well as the customers who are considering buying the products as well. The primary problem with reviews is that the text of reviews tend to be unstructured and noisy.

As a general rule, the opinion as a kind of problem needs to be structured in order to be understood (Cambria, Das, Bandyopadhyay, & Feraco, 2017). To assess the quality of the text, a set of operations must be performed on the product review text. For instance, review selection by rate or usefulness, review summaries, reviewer credibility detection, and entity resolution operation or text processing types that can be applied to the original review text. In most cases, these kinds of operations increase the usefulness of the reviews, and there are a variety of such operations like these in this wide area.

The scope of product review analysis is vast, encompassing sentiment analysis, opinion mining and review analysis systems, methodologies, and algorithms. Sentiment analysis is a strong topic that emerged under text analysis. A sentiment is an attitude, thought, or judgment that is prompted by feeling (such as joy, sadness or anger). The goal of sentiment analysis is to automate the extraction of meaningful and structured data from user-generated text (Kaufmann, 2017). A widely known form of sentiment analysis involves polarising the reviews into negative, positive, and neutral values. These attributes are called *polarities* (Cambria et al., 2017). A variety of studies have been performed on this subject, which involved grabbing reviews, detecting negative reviews, and verifying review usefulness, and analysing the review sentiment. For example, almost all the references used in this dissertation denotes some of the abovementioned topics as major concentrations in the studies.

Since the year 2000, many sentiment analysis frameworks have been introduced (Cambria et al., 2017), and this is still a highly active area of research (Chen & Sun, 2017).

As observed by this researcher, most of the topics studied in this domain so far have been about the efficiency and usefulness of the algorithms that are used for review analysis with the proposal for more robust techniques and solutions. The studies seem to have been considerably negligent about the software engineering concerns for such systems.

Concentrating on the techniques or algorithms of PRA[1] systems alone is not sufficient to address the overall quality of PRA systems. It is also essential to improve the

[1] The abbreviation PRA used in this dissertation denotes all software systems related to product review analysis.

practicality of these systems. Software quality metrics such as reusability and maintainability are essential to PRA systems just as they are with other systems. Although ample works have introduced the new architecture of PRA systems either explicitly (Flory, Kweku-Muata, Osei-Bryson, & Thomas, 2017; Tsirakis, Poulopoulos, Tsantilas, LTD, & Varlamis, 2015) or implicitly (Eirinaki, Pisal, & Singh, 2011; Petz et al., 2014), not much attention has been given to the quality attributes of the proposed architecture for PRA systems.

Having upfront architectural thinking is significant in this domain as well, as the system detailed design and implementation will follow the architectural concepts. Applying some new architectural styles would make the quality assurance (QA) easier. QA is essential to discuss the trade-off points in the system design. Similar to other software-intensive systems, applying quality-centric development to PRA systems can lead to a better separation between the layers or components and a more modularised and encapsulated system.

Quality attributes (QA) are significant in different ways. First of all, they are widely accepted by both the industry and academia. Secondly, quality achievement targets are very reliable measurands for software architecture success and effectiveness.

Studying software architecture in the existing systems would provides enough evidence for finding the gaps in the literature. In this regard, this research was aimed at understanding the latest architecture of PRA systems, identifying their limitations, and proposing some improvements. To evaluate the improvement, software quality attributes had to be evaluated.

Due to the variety of QAs available for software assessment, covering all of them would be a big hassle for a higher quality PRA system. Therefore, it made sense to concentrate on a few of them for this study to achieve some measurable improvements as compared to the latest works. This study values *maintainability* since there are different

channels for collecting reviews, as well as a variety of evolving techniques to analyse the gathered data. Therefore, contributing to the production of more maintainable PRA systems can be considered as an improvement. In particular, this study addresses maintainability from the point of view of coupling.

1.2 Problem Statement

A vast majority of researches and industry players in the online PRA domain have concentrated on ameliorating the current techniques for product review analysis, sentiment analysis, and opinion mining, such as increasing the accuracy of classifiers over the past decade. Considering the importance of these systems, the quality concerns of the systems are still relevant in the domain of PRA and need to be addressed.

Even though architecture was not the primary concern for a majority of the researchers, by chronologically reflecting on the proposed architectures for review analysis systems, the evolution of the architecture was made observable. The researchers have used the monolithic application (Flory et al., 2017), modular-based implementation (Petz et al., 2014), and indexing platform (Eirinaki et al., 2011) to increase document retrieval, clustering for high availability (Tsirakis et al., 2015), supervised machine learning, API-based implementation for reusability (Tsirakis et al., 2015), and, later, microservice-based implementation (Espinoza, Mendoza, & Ortega, 2018a).

Microservice architecture is about decoupling the system into reusable services aligned with business domain decomposition. The primary outcome of improving software architecture is to achieve or improve certain quality attributes, including the reusability, extendibility, maintainability, and scalability of the system. These attributes have been recognised as some of the advantages of microservice architecture by both academia and industry. Hence, microservice-based architecture forms one of the key architectural patterns for most of the QAs that this study targeted to improve. Some

existing works that have utilized microservices for PRA systems were evaluated the extent to which they have improved upon, taking microservice architecture into the account.

Exploring the limited researches on PRA systems, which have utilised microservice architecture to achieve quality attributes such as reusability and decoupling of system components, revealed that the PRA domain still welcomes studies focus on architecture, especially those on microservices.

A recent study that utilised microservices for a PRA system (Espinoza et al., 2018a) resulted in a product named *Viscovery*. The approach for the study was to expose the required review text processing methods and techniques as microservices. Viscovery's system provides application programming interfaces (APIs) for the exposed microservices, which is called *Novaviz* API gateway. Therefore, the most crucial decision for this architecture was to expose the libraries and algorithms as microservices. Viscovery's system design is a kind of improved architecture of the PRA system, leading to greater abstraction and decoupling between the components and utilities, as exposing the utilities as services always increases the reusability and level of modularisation. As a result, Viscovery is distinct from the previous PRA systems (that mentioned in the literature review), for focusing on extendibility and having reusable services. However, the following questions remain: Has Viscovery ~~a~~ fully utilised microservice architecture? To what extent does it comply with the patterns for microservice design and architecture?

The proposed architecture for Viscovery, is a hybrid architecture. This means that it uses both component and microservice-based architecture. The components invoke the required microservices by using the exposed API gateway. Some of the issues in this approach are as follows:

1- The components are coupled with microservices through the API gateway. The components invoke the microservices. The core elements, players, and initiators in the system are the components and not the microservices.

2- Some of the recommended peripheral elements in microservice-based architecture are missing from this system's architecture, such as registry and proxy. The registry is useful for monitoring microservices and keeps track of their availability. Proxy is a recommended component for high availability and load balancing.

As a result, there is still a gap in this research in terms of satisfying with microservice architecture patterns since some improvements can be made using more design principles recommended for microservice-based systems. Consequently, the claim made in this dissertation is that the abovementioned work does not fully utilize a microservice-based architecture, which the PRA domain can benefit from.

This research focused on solving the abovementioned issues to decrease the coupling of the microservices within the architecture and increase the benefits by using some recommended peripheral components in the microservice architecture and a systematic approach for microservice design, this research also looked at establishing clearer boundaries in the domain.

1.3 Research Questions

This research focuses on the most recent microservice-based PRA systems and aims at addressing the following questions:

1- What are the weaknesses and strengths of the existing PRA systems in terms of coupling ?

2- How can the current work be improved by applying microservice-based architecture, especially to reduce coupling?

3- What are the requirements for achieving low coupling in microservice-based PRA systems?

4- What technologies could be used to implement a PRA system based on the proposed microservice-based architecture?

5- What are the metrics for measuring the coupling of microservice-based PRA systems?

6- How can the coupling of microservice-based PRA systems based on the selected metrics be measured?

1.4 Research Objectives

The objectives of this research are as follows:

1- To propose a new microservice based architecture for PRA systems to reduce coupling.

2- To implement a prototype PRA system based on the proposed microservice-based architecture as a proof of concept.

3- To evaluate the coupling of the implemented prototype PRA system as the primary quality improvement measures against the existing systems.

1.5 Research Scope

This research aims at applying a microservice-based architecture in a prototype PRA system to improve an existing existing which is based on microservices (Viscovery), and to measure the coupling of the system. This research, however, did not attempt to find a novel strategy for improving the efficiency of existing PRA methods, techniques, and algorithms, or refining the reviews and visualisation,.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter presents background information on the review analysis and related works on PRA systems in the literature. The following section is dedicated to a brief introduction to opinion mining and sentiment analysis as well as their significance. After this, the significance of the architecture in PRA systems is explained; of all the software architecture style, microservice architecture is considered for the explanation. Subsequently, the existing PRA software architectures are discussed, and their advantages and disadvantages are considered. Finally, the existing gap in the literature will be discussed further.

2.2 Product Review Analysis for E-Commerce Platforms

In the competitive e-commerce world, customer feedback about the product or service is crucial. The value and significance of Electronic Word of Mouth (E-WOM) or User Generated Content (UGC) are incomparable, as opposed to traditional surveys or customer's opinion collection, as E-WOM is much more critical. The introduction of E-WOMs, and the platforms for mining the same negates the need for surveys, opinion polls, asking friends and family members, and groups. As a piece of evidence, it is interesting that Twitter comments have been cited as some of the most influential factors of branding images (Robson, Farshid, Bredican, & Humphrey, 2013). In comparison with traditional opinions, E-WOMs will last longer and never vanish (Robson et al., 2013).

E-WOM is, in essence, opinionated, unstructured, subjective, massive, and difficult to interpret, so collecting, refining, and analysing the content is crucial. Unlike objective statements that can be proven wrong or right, opinions, and especially E-WOM in social media, are subjective. For instance, "This notebook's got battery" is objective, while "This notebook's battery is the best in the world" is subjective. E-WOMs generates massive amounts of data that get buried in the unstructured text. This sort of unstructured

data is not limited to the product reviews on e-commerce web sites and can be about political issues, branding, marketing campaigns, and more.

E-commerce studies indicate that there is a direct relationship between an online review and online purchase. The collection of enriched representational opinions is not only useful for producers to understand the market trends and evaluate customer satisfaction levels but also to trust and verify previous customer viewpoints about the products and services. It is also vital for the merchants or service providers to determine the product features that customers like more, as well as their general feeling and emotion toward the company's brand or a specific product. Therefore, opinion mining is a highly important concern in this domain.

Opinion mining involves a set of tasks for enriching opinions, including the detection of opinion holders, what is the topic, what is the context, and what is the content. They are good decision support drivers for understanding people's preferences to serve them better. Opinion mining also helps advertise the product or service to the right people that is called *targeted advertising* (Kaufmann, 2017). Social sciences and market research take advantage of humans as sensors and aggregated opinions. (Kaufmann, 2017)

There are three different categories of sentiment analysis techniques: *knowledge-base, statistical, and hybrid* (Kaufmann, 2017). The first category primarily relies on knowledge of analysing and classifying. Statistical techniques use support vector machines and the machine-learning approach, but these techniques are typically weaker than those that are analytical (Kaufmann, 2017). The hybrid approach, as the name denotes, utilises a combination of both methods.

On the one hand, due to variety, velocity, volume and veracity of reviews or opinions on different Web 2.0 channels, such as social media, websites, and blogs, it is hard to catch the reviews on time. On the other hand, if merchants or service providers do not

take product reviews into consideration, they will miss the market trend of worldwide business competitions and marketing challenges.

Within the industry, it can be observed that some companies such as IBM, Oracle, SAS, SentiNet, and Luminoso have developed some commercial off-the-shelf tools to detect and analyse a customer's emotions and mood. In the academic sphere, a large number of researches concerning this subject indicate that opinion mining has a vital role in evaluating business success, political achievements, and social trends.

Further, reviewing the studies in this area shows that even after two decades, many research works are still in progress. Figure 2.1 shows the Google Scholar results for sentiment analysis. (Chen & Sun, 2017)

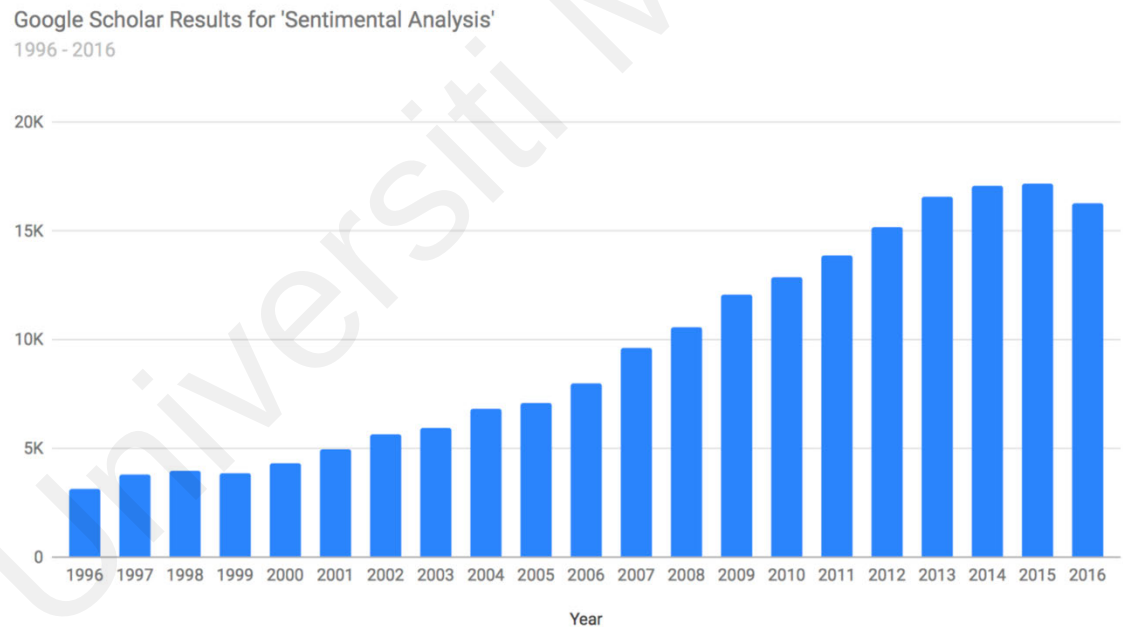

Figure 2.1 Research papers relating to 'Sentiment Analysis' (Chen & Sun, 2017)

The majority of the research in this area is related to marketing, management, and computer science techniques. Considering some of the works since 2004, the research subjects in this domain are about identifying the *usefulness of reviews*, *feature extraction*, *sarcasm*(detecting spam and fake opinions), *entity recognition* (detecting opinion holder), and *context identification*.

However, it is necessary to use different approaches for opinion mining, as one approach may compensate for the cons of another methods (Galin, 2018). Using a variety of approaches for sentiment mining and analysis emphasises the importance of maintainability and flexibility of such systems. The growing volume of reviews and various media also indicates the need for scalability. Therefore, there is a need to consider quality as an indispensable factor for the success of PRA systems.

Quality assurance (QA) is in relation to architecture. The aim of QA is meeting the written requirements for the customer or stakeholder's satisfaction (Galin, 2018). The QA perspective can be seen explicitly or implicitly in most of the existing works, even though it was not the primary target for most of the PRA researches.

At the very beginning, when research in this field began, architecture was not the primary concern. Instead, solving review analysis problems and issues was the primary focus. However, later researches have studied the subjects of the platform, architecture, and components required in PRA systems.

One aspect that makes PRA systems useful and flexible is the systems' architecture. Considering the existing literature from a design or architectural perspective was one of the targets for this study. The main goal was to use the existing platform as a benchmark for the application of QA and to make improvements.

In the next section, some of the outcomes for the architecture exploration activity in the existing PRA systems will be highlighted to identify the research gap and propose an improved architecture by applying the latest architectural pattern for E-WOM collection and analysis in order to realise a system with better quality. Achieving a high-quality system also meant complying with some quality factors; this will be introduced in a later Section 2.5.3.

2.3 Exlporing the Existing Architectures of PRA Systems

There are three essential dimensions for a successful application: architecture, organisation, and process. Architecture plays a vital role in every software system. Subramanian (2020) provided a simple and interesting definition for architecture: "A shared knowledge about essential components of the system". As the size of the applications increases, the design and forethought about the applications' architecture, which includes software components and communications among the components, becomes much more critical in the software development lifecycle. Architecture matters as it affects the quality attributes[2] of software systems.

Due to the variety and volume of the resources for UGCs and E-WOM as well as the diversity of the opinion mining methods, algorithms, and integration requirements with external systems (such as CRM systems), a PRA system could become a large and complex application or a system of systems. Like other systems, PRA systems implicitly or explicitly come with an architecture. If the design, architecture and quality attributes are dismissed or underestimated when developing a PRA system, it will not lead to a high-quality PRA system that brings satisfaction to the stakeholders.

Based on the existing literature, the evolution of PRA systems would be worthwhile to study. Further, the architectures are not explicitly described or missing in the existing works. Therefore, picking up one of the latest, most adequate architectures and working on it further can pave the way for an improved software architecture version that serves the domain of RPA.

[2] In some software architecture references, the software quality attributes are also called *ilittes* and include, among others, scalability, maintainability and usability.

This study explored implicit or explicit architectures in the existing works based on the following criteria:

1- Extracting the documented or undocumented architecture of the PRA systems.

2- Looking for main building blocks or main modules.

3- Looking for the architectural style or patterns used in the systems.

4- Exploring the improvement in terms of modularity of different PRA systems over time.

5- Looking for creating decoupled components or reusable services.

Exploring the research background showed that, the candidate components for opinion mining systems were introduced back in 2004 (Hu & Liu, 2004). The authors suggested some of the main components, such *as POS tagging*, the *frequent feature generation*, and *opinion extraction*, and also concentrated on *bag-of-features*, *context-identification, and sarcasm*.

Later, the other researchers concentrated on proposing algorithms for extracting sentiments. They employed the algorithms in a search engine for *feature selection and classification*. (Eirinaki et al., 2011). Feature extraction was the primary goal in these works. Features extracted is vital, as what the customers like or dislike can be predicted based on a subset of features from a large pool of features. The authors discussed the architecture very little and mainly focused on the *High Adjective Count* algorithms for feature extraction . (Eirinaki et al., 2011)

Since, one of the main architectural concerns is to identify the main components or building blocks of the system, finding the components required for opinion mining is a

necessary requirement, as stated before with regard to the research focused on feature extraction. The opinion mining components identified from the PRA system proposed by Eirinaki, Pisal, and Singh (2011) are shown in Figure 2.2. Data pre-processing, as the name implies, processes the original review's text. For example, if the review is of a specific product, the pre-processor separates that as a new file. An opinion mining engine is embedded as a Part-of-speech (POS) tagger to discriminate different words' grammatical roles within the review text, i.e. noun, verb, adjective, adverb, and so on. The opinion ranking component assigns a score to the opinion based on the direction of the opinions (Tsirakis et al.) for each feature. Finally, the indexing component indexes the opinions based on the features for faster retrieval through the user interface.

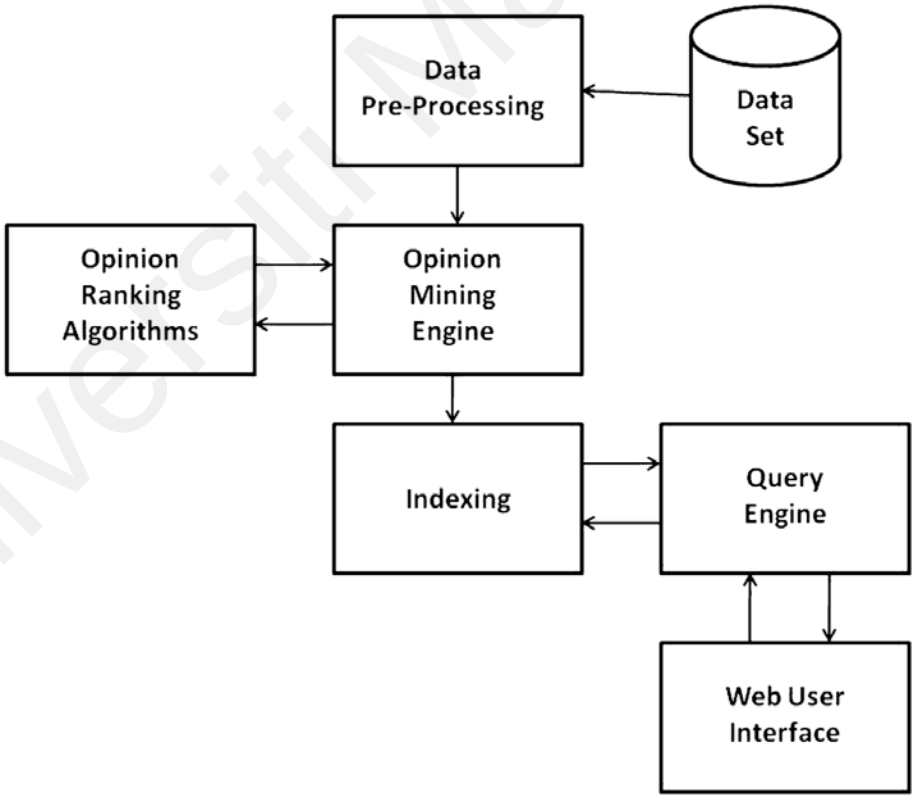

Figure 2.2: Feature-based opinion mining (Eirinaki et al., 2011)

Robson etc all. (2011) proposed a *Leximancer* software package to identify concepts, rather than keywords, in the text. One of the components of Leximancer is a kind of a natural language processor engine. The authors used the links between the extracted concepts to identify contradicting reviews. The output of Leximencer is a concept map. The following Figure shows a sample concept map. However, they didn't explain how Leximancer can be used. The other components required to make up a PRA system were not mentioned either. Thus, their study was concerned with text processing, and they used Leximancer only as a tool for the same. They also did not provide a clear structure of the proposed system for mining meaningful words (concepts) from the unstructured text.

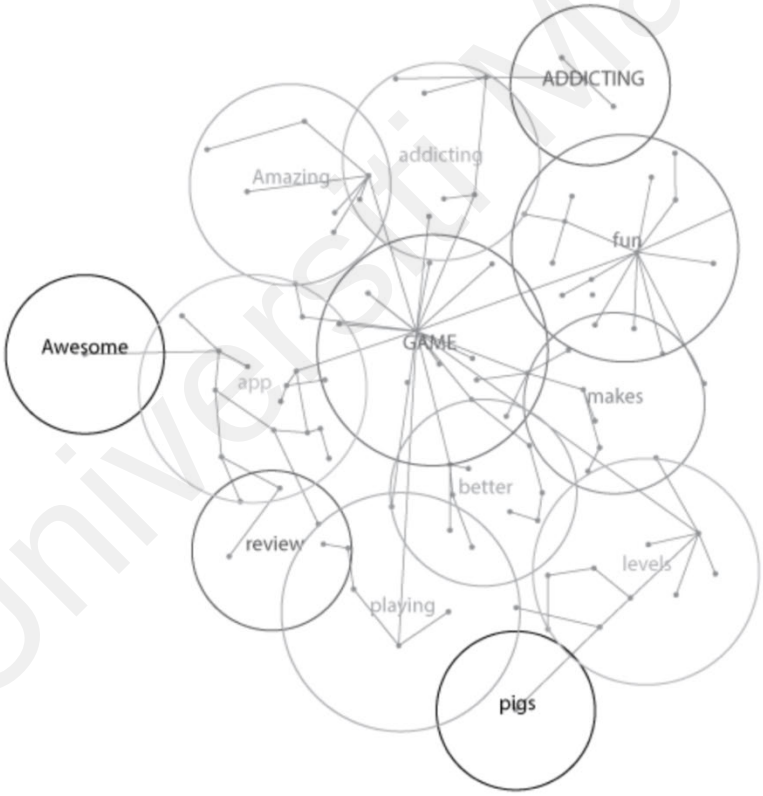

Figure 2.3: Concept Map is for *Angry Bird, Fruit Ninja, Tiny Wings* and *Cut the Rope* (Robson et al., 2013)

Petz et al. (2014) evaluated the discrepancies between different social media channels. They also identified different algorithms for text processing on social media based on NLP and investigated the effectiveness of the algorithms. Based on the ancestor works, they introduced two categories for opinion analysis*:* (i) *lexicon or dictionary-based mining* and (Petz et al.) (ii) *machine learning approach*. Machine learning itself can be categorised as supervised, unsupervised, as well as those with other approaches. One of the exciting results from this work was the system's consideration of grammatically incorrect texts in the processing. Figure 2.4 shows the grammatically incorrect sentences found on different channels. However, from an architectural standpoint, it is not clear how the different algorithms were implemented and evaluated.

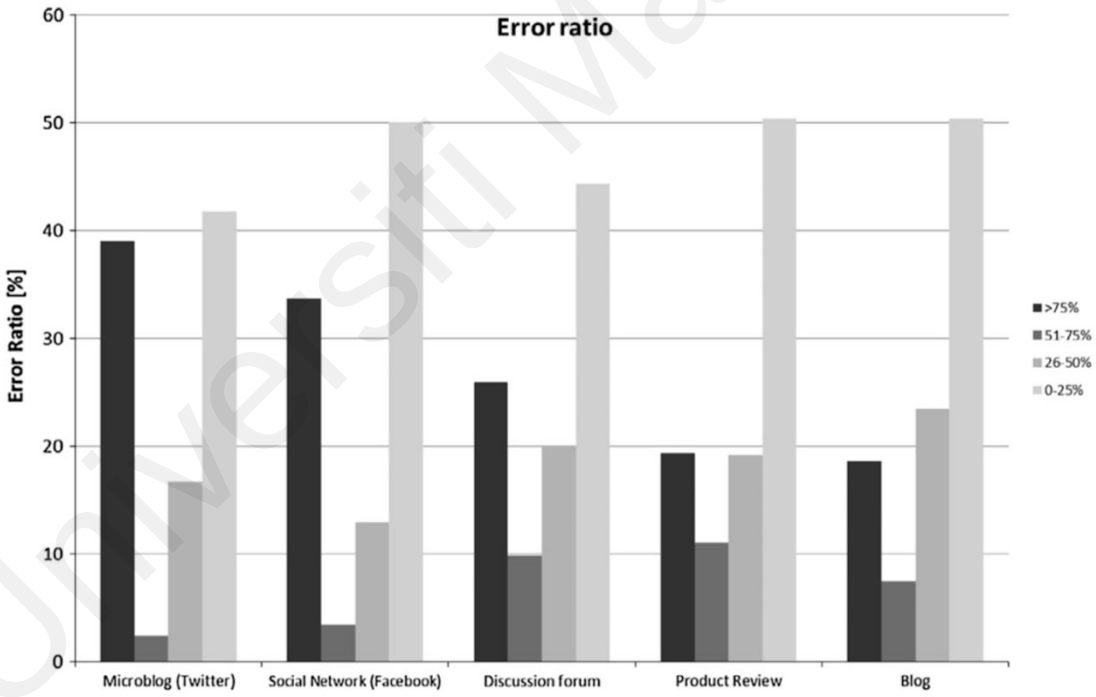

Figure 2.4: Grammatical error ration in different channels (Petz et al., 2014)

Cristin Bucur (2015) also recommended a modular architecture for sentiment analysis systems, consisting of four modules: *Acquisition, Storage, Statistics, and Analysis*. The platform collects data and stores, classifies and centralises the results. For this purpose, the acquisition module collects reviews from different sources, requiring the storage

module to support multiple high-speed read queries. This specification is normally supported by NoSQL databases. The analysis module performs sentiment analysis and classification functions. The statistics module is responsible for the visualisation and accuracy measurements by utilising some of the proposed algorithms.

Even though the described system architecture was much clearer than the ones in previous works, the study's focus was on classification and scoring algorithms. However, a high dependency and interaction levels between the proposed modules can be observed. Thus, the system modules were not loosely-coupled. Figure 2.5 represents this mentioned dependency.

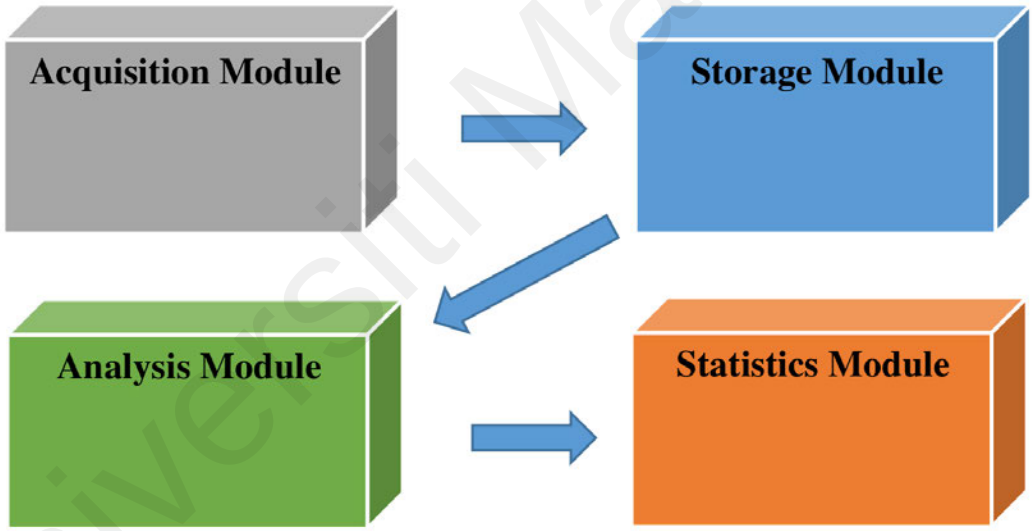

Figure 2.5: Modular architecture for a PRA system (Bucur, 2015)

Another work with architectural components was carried out by Tsirakis, et al. (2015), who proposed a platform called *PaloPro* with the following features: real-time opinion mining, source prioritisation, probabilistic language support, feature-based Indexing, and crawling. This is one of the significant works in which architectural components and infrastructure-related details have been emphasised. The architecture depicted in Figure 2.5 shows the physical view for the system; Unfortunately, the

conceptual view was unavailable to extract more information. The proposed architecture was module-based, and the modules such as crawling, feed aggregation, clustering, and multi-document summarisation had been incorporated (Tsirakis et al., 2015).

In 2017, the PaloPro team described more about the system's component-based architecture further, which is a kind of (Tsirakis et al., 2015 & Varlamis, 2017b). The major components were as follows:

Similar to previous works in the field, PaloPro also includes a feature for monitoring people, companies, and other entities on social media. The underlying linguistic module has 87% accuracy for polarity detection and named-entity resolution. Polarity refers to the ratio between the number of positive words and the number of negative words. The crawler collects reviews from different sources, including Facebook, Twitter, video comments, and normal websites and blogs. The *content aggregator* filters the data upon collection. Based on the frequency of visits, the *crawler components* then prioritises the resources. *Spam detection filters* can also be implemented in multiple layers to refine the reviews.

PaloPro also includes a component for *visualisation* using dashboards called *Workspace*, which allows the user to measure an entity's (brand, company, product, or person) reputation based on the selected keywords and attributes. The process follow (the functionality) for PaoloPro is similar to any other PRA system, starting with raw data and leading to business knowledge through the functions of data acquisition and recording, information extraction and cleaning, data integration, modelling and analysis, and interpretation and visualisation.

Tsirakis et al. (2017) used both SQL and No-SQL databases for indexing and fast data retrieval, both related to the collection and analysis components of PaloPro. They

provided an API for visualisation and managed to launch the platform in another country within a few months. As a crucial decision for the system architecture, they used stream data processing pipelines. By 2017, PaloPro could already support other languages and cross-country social media. It was a significant achievement in terms of flexibility and portability. Nevertheless, there are still questions yet to be answered: How easy is it to change the features related to the social media channels? How easy is it to change and deploy the components? To what extent is the system modularised? Did the developers assess the architecture based on QAs or, in another word, *illity* measurands?

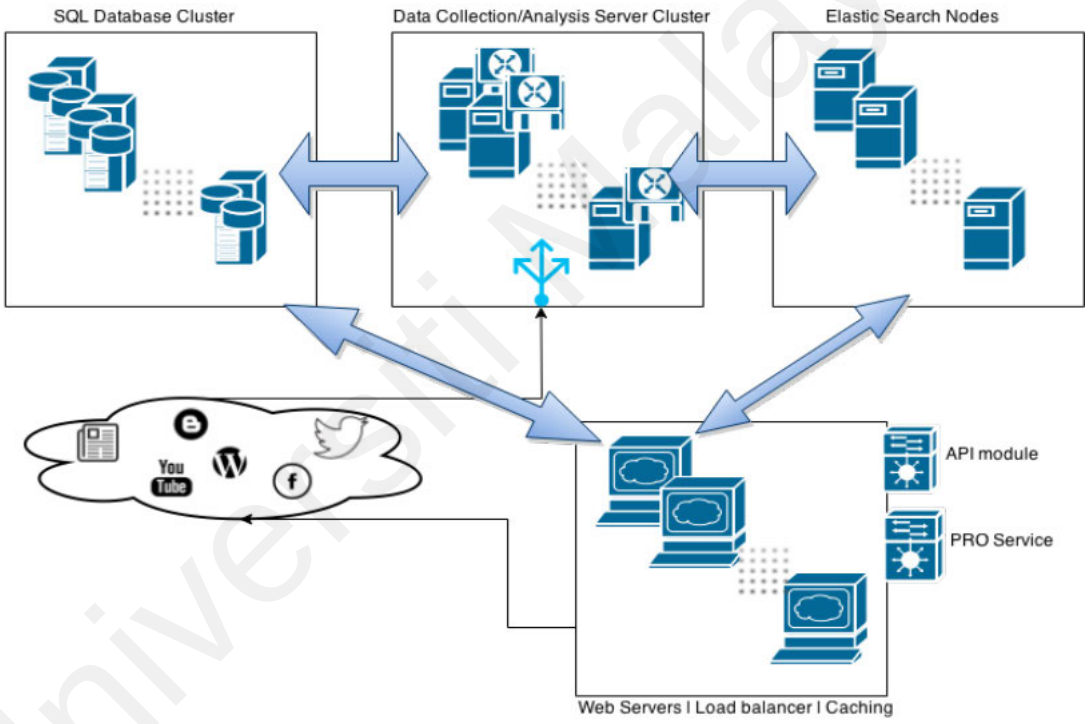

Figure 2.6: Paolo Pro physical view (Tsirakis, 2017)

A general-purpose service-oriented architecture for the *IUSR Analyzer* was proposed by Flory etc al. (2017). They incorporated three main building blocks in the architecture: Pre-Filtering, User Interface (client), and Back End (server). With modules in each block. The Pre-Filtering block consists of two modules: Review Post Sensor and Review Spam Detector. The user interface and the back-end building blocks are composed of six

interrelated modules, each module contains one or two components. The authors' work primarily concerned spam detection and review quality to ensure that the text makes sense to the customer. Figure 2.7 shows the architecture of the IUSR Analyser.

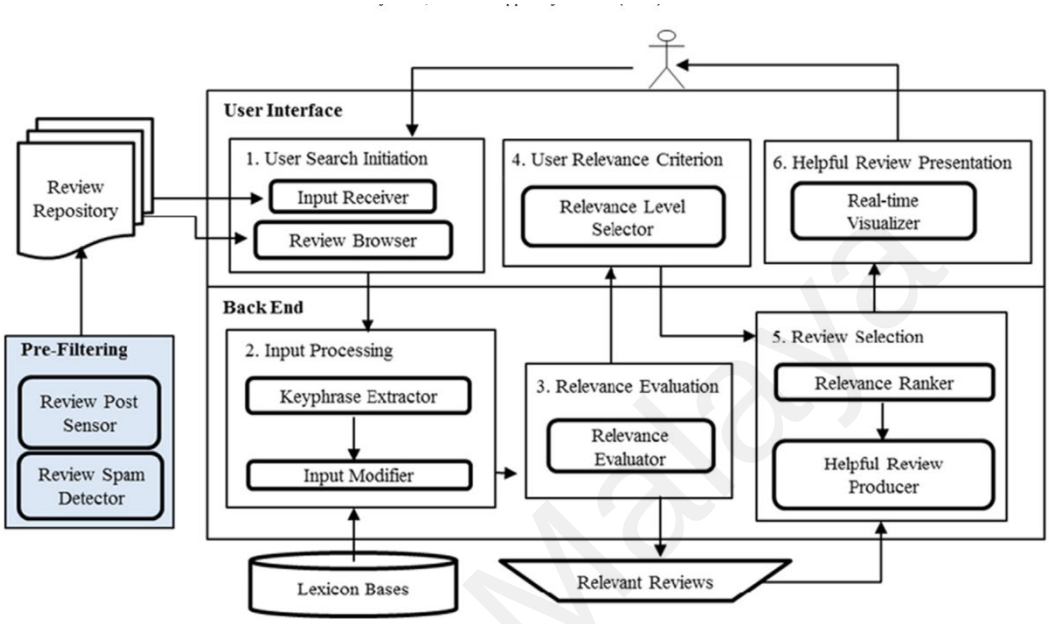

Figure 2.7: Architecture of IUSR Analyzer (Flory et al., 2017)

The most recent related work reviewed was Viscovery (Espinoza, Mendoza, & Ortega, 2018b), a platform for trend tracking. The target for this work was to track trends using dynamic topic modelling, which allows the evolution of the topics to be followed over time. As a document would contain topics, and each topic would contain some words, The Vador Lexicon output can be used to calculate the polarity score based on the word, topic, and document level.

The authors' main contribution is that they structured different algorithms as microservices. The core components of the proposed architecture are Data Injector, Data Pre-processor, Data Processor, and Indexer. They decoupled the visualisation components (Kibana and DFR Browser) from the back-end by implementing the API

gateway. The data Injector is a crawler, while the data pre-processor filters and clears sentences. The data processor, which is the main component, performs sentiment and score calculation, and the indexer stores the final data in an elastic search repository. All these components communicate with each other through the Novaviz gateway.

Even though, from the architectural perspective, this system shows a significant improvement in terms of coupling, further investigations into this work revealed that the components can be made much more cohesive. The microservices synchronously communicate (request/response) through the API gateway. However, synchronous communication has some drawbacks. If one service does not reply, the caller service will be blocked for it, which could lead to the outage of the entire system. Therefore, one of the improvements that can be made for low-coupling is implementing asynchronous communications rather than having to call another service and wait for the response. Figure 2.8 shows Viscovery's architecture.

The main issue here is related to the question, can Viscovery be considered as a microservice-based application? By matching the microservice architecture patterns with Viscovery's implementation, the following notes were observed.

Of most important one was that Viscovery just exposed utilities and algorithms as microservices. The system process is still carried on by the components. The API gateway does not necessarily route requests or balance the requests between different services, and it provides a single point of access for the exposed microservices.

In terms of availability and scalability, some peripheral recommended components for microservice architecture can be employed in Viscovery's structure. For instance, adding a service registry for service discovery and registration would

improve the architecture. Likewise, load balancers would increase system availability. These elements were never discussed in the author's work.

As a result, it turns out that this study did not fully utilise the microservice architectural style.

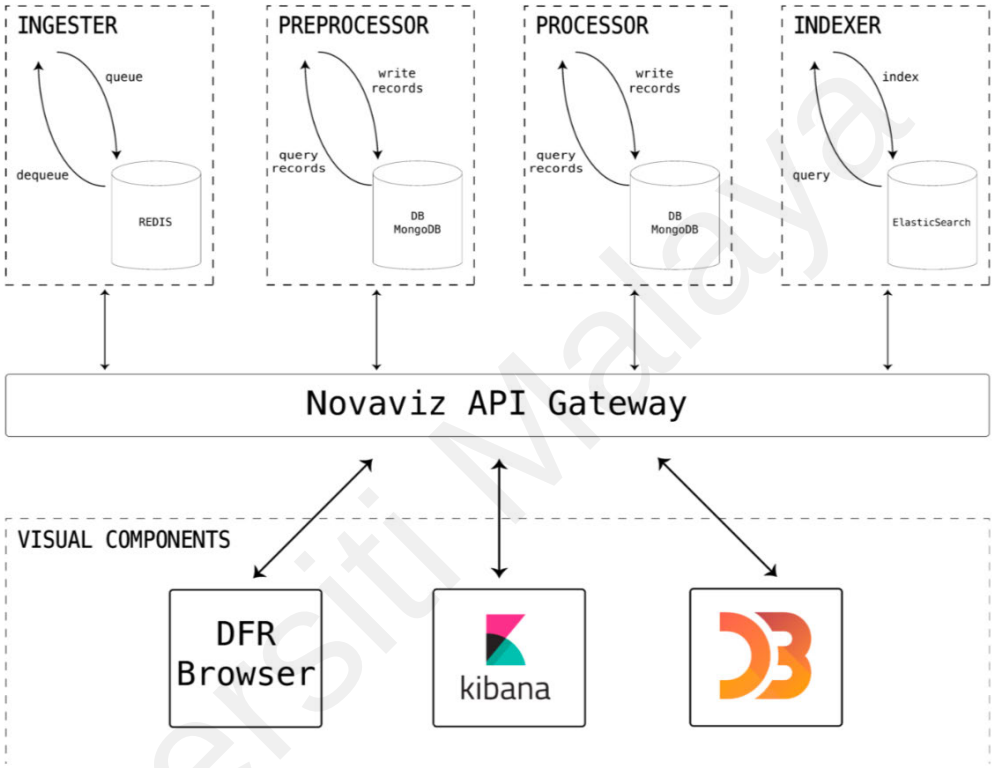

Figure 2.8: Viscovery - a microservice-based platform (Espinoza et al., 2018b)

As previously mentioned, the study reported that most of the algorithms were exposed as microservices. Therefore, with proper decomposition, the microservices could be refactored in an efficient way to align them with the domain capacity. For example, there is no clear microservice breakdown in the system, and based on Figure 2.8, the system is a component-based one. The structure is composed of four components; injector, pre-processor, indexer, and processor. The authors divided the entire domain into four main

categories and implemented some functions related to sentiment analysis and review processing as microservices.

It is worthwhile to note that the primary aim for the Viscovery was not architecture but the visualisation of on opinion topics using the DFR browser. Thus, a more thorough study of the architecture and the platform of PRA systems is relevant and required. Even though the authors mentioned the presence of the microservices, the extent to which they utilised the microservice architectural patterns remains unclear.

Petz (2019) highlighted the general process for opinion mining, focusing on its methods and techniques.

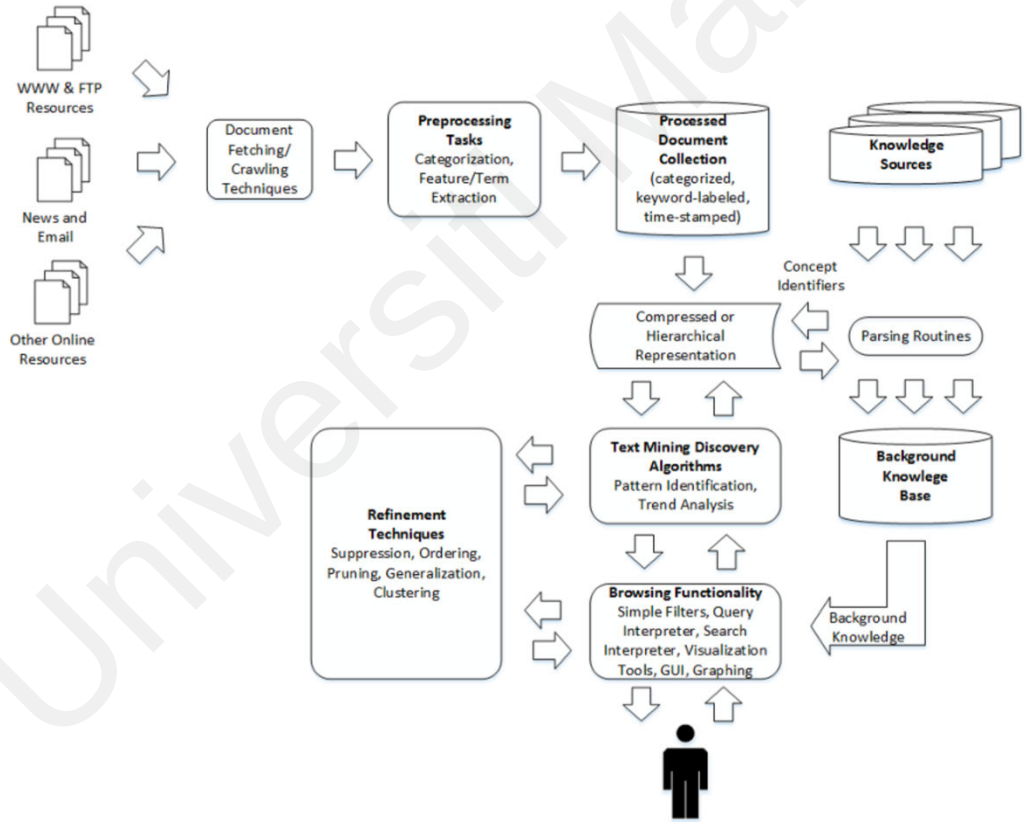

Figure 2.9 Main components for knowledge discovery through data mining (Guo et al., 2011, as cited in Petz, 2019)

Figure 2.9 illustrates a proposal of components for knowledge discovery through data mining (Guo et al., 2011, as cited in Petz, 2019). The identified components include document fetching, preprocessing, processing, which are common for opinion mining

systems. Even though the purpose of the above study was retrieving patterns and knowledge out of the processed review or opinion text, the identified main building blocks serve as a reference for this study.

It is noted that the main building blocks for the general opinion mining system is similar in a more recent architectures (Bahatia, Chaudhary, & Day, 2020), as shown in Figure 2.10. Figure 2.10 illustrates retrieval, filtering (identification) and processing (classification) and summarisation as main components. This work also did not focus on architectural improvements for opinion mining or product review analysis systems, it's gross-level architectural components confirms the earlier proposal by Guo et al. (cited in Petz, 2019).

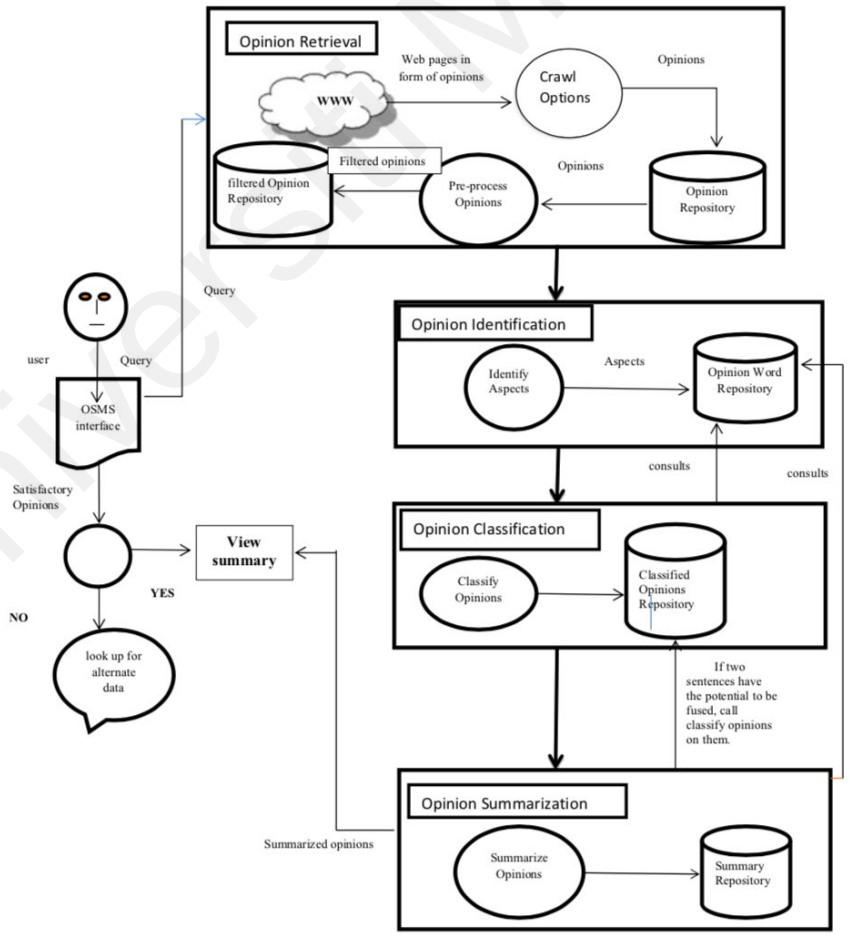

Figure 2.10 Block digram for an opinion score mining system(Bahatia, Chaudhary, & Day, 2020)

2.4 Review of the Existing Architecture

Despite the significant evolutionary progress in the PRA system architectures reported explicitly or implicitly in the existing works, there is still room to improve for the architecture of the PRA systems. This would bring many advantages to PRA systems by increasing its quality and adding new scientific research outcomes to the system.

Table 2.1 summarises the architectures of the existing PRA systems in chronological order. This table shows the main architectural style for of the systems and the extent to which the modules or components depend on each other (coupling).

Table 2.1 Architecture of existing PRA systems

Year	Study	Main acrhitecture	Highlighted problem in terms of coupling
2011	Feature-based opinion mining (Eirinaki et al., 2011)	Modular	Highly dependent modules
2013	Five-star review (Robson et al., 2013)	No clear structure	*Highly dependent on Leximancer, no clear and well-documented architecture*
2014	Algorithms to carry out text preprocessing (Petz et al., 2014)	No clear structure	*No clear and well-documented architecture*
2015	Modular architecture for a PRA system (Bucur, 2015)	Modular	Highly dependent modules
2017	Paolo Pro (Tsirakis, 2017)	*Component based*	*Highly dependent compoent*
2017	IUSR Analyzer (Flory et al., 2017)	*Component based*	*Dependent components*

2019	Viscovery : A Platform for Trend Tracking in Opinion Forums (Espinoza et al., 2018b)	MicroService	*Not fully utilising microservice architecture*
2019	Opinion mining in Web 2.0 (petz,2019)	No clear structure	*No clear and well-documented architecture*
2020	Opinion score mining system (Bahatia, Chaudhary, & Day, 2020)	No clear structure, focus more on the process and the pipeline	*No clear and well-documented architecture*

The following items represent the major requirements of PRA systems observed in the existing PRA systems:

1- Ability to adapt to the new channels as the sources for extracting the product reviews are growing.
2- Ability to apply and change the review processing algorithms and techniques for incorporating and upgrading new techniques such as machine learning and big data processing.
3- Ability to adapt and run the platform for different languages and different countries.
4- Ability to maintain the system as PRA systems grow fast in relation to the volume of the reviews and increasing number of products.
5- Availability of the main system components to grab and analyse the reviews.

By carefully considering the above-mentioned requirements, an important question to ask is whether the existing PRA system architectures fulfil the requirements? Requirements 1 and 2 denotes for agility to adapt to the existing and emerging

requirements. This requires loosely-coupled system components and well-defined breakdown in the system components to change and deploy it fast as the PRA domain is changing.

Requirements 3 and 4 relate to system scalability as the review volume, and languages and countries to be supported are increasing. The methods and techniques for extracting, analysing, keeping, and indexing are also growing in the evolutionary cycle. For this reason, PRA systems need to adopt the latest technologies such as microservice architecture as the systems are becoming large and complex. In addition, Requirements 5 relates to system availability.

Based on the analysis, this study proposes microservice-based architecture as an ideal solution for fulfilling Requirements 1-4. The following sections describe Microservice architecture aims to address the above-mentioned requirements.

Microservice architecture is a suitable architecture for modularising PRA systems and decomposing them into interrelated components. Loosely coupled, independent, deployable, and autonomous microservices can expedite the development time and the platform adaption for the business cases.

2.5 PRA and Microservice Architecture

This section answers the two main questions below:

a) Why microservice architecture improves PRA system design?

b) How to measure the improvement?

2.5.1 Microservice Architecture

Microservice architecture is an architectural style based on service-oriented architecture (SOA). There are some differences between microservice architecture and

SOA. First, microservices are much smaller than the typically large and orchestrated services in SOA. Secondly, SOA normally uses smart pipelines such as Enterprise Service Buses (ESBs) for communications between services, but microservices communicate using dumb message brokers (Rihcardson, 2019). Despite this, SOA in microservice ecosystem is moving to the endpoints (services itself) and brokers or pipes are just moving the messages. (Lewis & Fowler, 2014) Generally, microservices can be considered as a subset of SOA that brings further value to the whole idea. Both SOA and microservice architecture aim to improve and facilitate modifiability, deployment, operations, and increased flexibility through modules such as autonomous and independently deployable services (Francesco, 2017).

The older architectural style is monolithic. Microservices are usually in contrast to the traditional monoliths, which turns out to be a big hassle since it is hard to keep the monoliths modular and maintainable. Refactoring them is also a big hassle because of tangled dependencies. The problem with large, old monolithic style applications is called the *monolithic hell*. Generally, every successful application has a habit of growing. As the system evolves or grows, modularity gets eroded (Tyszberowicz, Heinrich, Liu, & Liu, 2018). The sizeable monolithic system will eventually be the same as a big ball of mud (Rihcardson, 2019). This means that the software lacks architectural thinking. Further, when dealing with large or complex monolithic applications, the illusion of being agile in development eventually goes away.

Big companies, such as Amazon started migrating away from monolith since 2002. They later obtained impressive results, able to deploy changes every 11.6 seconds in their production in 2011 in a way that users could not feel any outage in the system. (Rihcardson, 2019).

A recent study shows significant interest in the use of microservice architecture (Granchelli et al., 2017) to cope with the issues of monolithic systems while providing a solution of lighter weight than SOA. A microservice-based system is composed of autonomous and independent microservices aligned with a sector, function or capacity in a business. *Componentisation* takes place in the microservices but in a way that components are services and not libraries. Components are replaceable and independently upgradable, and like services, they are also independently deployable. When a component is inside of a monolithic application, it would be harder to upgrade; but when it is in a microservice, the situation is different. The difference is that it is possible to independently deploy the microservices, while libraries are not autonomous and independent (Lewis & Fowler, 2014). Thus, each microservice could run in its own process.

A crucial point of note about microservice architecture is that each service has a business model, database, and specific behavior or business capability. This is called a Bounded Context (Tyszberowicz et al., 2018). The Bounded Context is one of the most substantial concepts of a microservice, as it plays an active role in achieving loosely coupled and highly cohesive services. It means that each microservice has it is own business model, including its own data, logic, and behavior. Therefore, a microservice, as a module, has an impermeable border that other micro-services usually cannot bypass. In microservices-based systems, services communicate with each other through the API and they do not use a unique or huge database for the integration.

Another benefit of microservices is that different teams can work on separated microservices, such that small teams can own small micro-services[3]. There is strong conciseness through the idea that, rather than having a big development team, the organisation can split the development into a team of teams (Rihcardson, 2019). It is hard to try and adopt different technologies or languages with the monolithic architecture, but it is easily achievable with the microservice style. Microservice architecture brings some specific requirements to the underlying infrastructure, especially with the use of lightweight containerisation platforms such as Docker, to facilitate deployment, scalability, and resilience.

Microservices are loosely coupled, and the communications between them take place via API or using event-based architecture. The event-driven approach can solve some of the complexity issues in the micro-service architecture. It will be discussed in Section 2.6 that the. Coupling in software design will be discussed in more detail in Section 2.5.5

Even though the benefits of microservice architecture are numerous and undeniable, achieving microservice architecture is a challenge in itself. One of the drawbacks of microservice architecture is its increasing complexity. For instance, developers need to be mindful of a partial system failure; When a microservices tries to communicate with another, the called microservice may not be able to respond within a reasonable period. Designing, testing, deploying and operating this kind of system is not easy either. Nevertheless, there are some solutions for all these problems unlike the complex issues of the monolithic applications. Besides, the use of microservice architecture increases

[3] They call it two pizzas as the team is small enough to be fed by two pizzas.

time-to-market, developer productivity, and scalability (Killalea, 2016). As a result, the benefits of microservice architecture worth and outweigh all the drawbacks of the complexity of the big applications.

2.5.2 Using Microservice Architecture for Developing PRA Systems

As shown in Table 2.1 architecture of PRA systems changes from monolithic to modular over time. The PRA domain need to get benefit of being service based on moving toward microservice acrhirtecture. The use of microservice architecture in the product review analysis domain is appropriate, which includes cloud-native architecture, and PRA systems mostly need to run on the cloud platforms as they are dealing with online resources. Therefore, based on the massive volume of reviews and the variety of resources and unstructured text, hosting the PRA systems on the cloud is the recommended choice, and will help with flexibility, scalability, and resilience.

As previously presented in Section2.3, microservice architecture has recently been applied in PRA systems. (Espinoza et al., 2018b). Even for existing systems based on monolithic architecture, it is also favourable to decompose the systems into service-based systems to better cope with the general requirement of PRA systems. Certain migration patterns are also recommended for migration from monolithic into micro-service architecture (Balalaie, Heydarnoori, Jamshidi, Tamburri, & Lynn, 2018) such as decomposing monolith based on data ownership, service registry, load balancer, and configuration server. Some of these patterns have been employed In the proposed architecture for the PRA system in this study.

Nevertheless, the existing PRA systems presented earlier in this dissertation (in Section 2.3) shows that microservice architecture has been newly adapted to such systems and has not been fully utilised.For instance, Viscovery (Espinoza et al.) should take advantage of additional components such as the service registry and API gateway to

maximise the microservice architecture's capabilities. Communications between the microservices would be of lighter weight and services would be decoupled further with the addition of these components.

2.5.3 Software Quality Metrics

According to IEEE Software (IEEE Std.730-2014)quality depends on the extent to which a software fulfils the user's requirements (Galin, 2018). Softwares of poor quality are hard to debug and maintain and easily become outdated and obsolete with time. Retrofitting high qualities to existing software systems is always tricky and requires much effort.

Software quality models date back to the 1970s, such as the Boehm and McCall models. There are some taxonomies and somehow the same factors for the quality. Both of these models classify quality attributes into a hierarchical model that includes categories and sub-categories (Galin, 2018). MacCall introduced 11 main quality attributes and grouped them into categories based on product operation, product version, and product transition as shown in Table 2.2.

Table 2.2: MacCall's quality model categories based on the requirements

Requirement	Quality attribute
Product operation	Correctness, Reliability, Efficiency, Integrity, Usability
Product revision	Maintainability, Flexibility, Testability, Efficiency
Product transition	Portability, Reusability, Interoperability

One of the quality model standards is ISO25010 (Galin, 2018). The quality characteristics given in ISO/IEC 25010 are functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability. QAs selection depends on the context of the system, including the domain, requirements, and

architecture. Microservice architecture has it is own role in achieving quality attributes. *Maintainability* is a primary attribute in a microservice-based system, as it has a direct relationship with coupling. As a general fact, loosely coupled services are easy to maintain.

Maintainability refers to the level of ease with which users and maintainers to detect the reasons for software failure and to correctly verify that the software is functionally working . Likewise, how much easier it would be for the developers to detect faults and improve upon or adapt the system is also related to maintainability. For example, a typical maintenance requirement is to determine the size of a module or program. If the module boundaries for a system are not clearly specified, the system would be hard to maintain it. The maintainability sub-characteristic based on McCall's quality model and the ISO/IEC 25010 are listed in Table 2.3.

Table 2.3: Maintainability Sub-Characteristics Based on McCall and ISO25010

McCall's	ISO/IEC 25010
Simplicity, modularity, self-descriptiveness coding, documentation guidelines, compliance (consistency)	Modularity, reusability, analyzability, modifiability, testability

On the other hand, maintainability can also be related to the ability to deploy and easily deliver software features. Maintainable software will facilitate *DevOps,* a prominent topic in the industry, which refers to a set of practices for quick, frequent, and reliable delivery of the product. (Rihcardson, 2019). Considering sub-characteristics of maintainability such as testability and modularity, the software's ability to deploy and self-describe within the microservice architecture may contribute to some improvements in this vital operation for the software development life cycle. In essence, one of the success factors in micro service development is infrastructure automation, including

continuous delivery, testing, and monitoring facilities. Covering this topic is outside the scope for this study, but it is worth noting that splitting a system into small deployable components would make it easier to manage the entire system's availability, but of course it requires automation infrastructure.

Another quality or sub-characteristic related to this work is *reusability*, which is the ability to develop reusable modules or incorporate already developed modules into a new system. Reusability is an essential characteristic of microservices is reusability. As microservices have already been tested and the failure cases have also have been detected by former developers, and microservices are well-defined enough to integrate them into existing applications using API.

2.5.4 Maintainability Model for Microservices

As mentioned in the previous section, the International Organization for Standardization proposed the quality model ISO/IEC 25010, but practical guides are missing. In addition to the quality model, Some other researchers tried to elaborate further on practical approaches to realise a quality model. They used quality metrics for the object-oriented design of SOA, but the proposed methods were too complex (Bogner, Wagner, & Zimmermann, 2017).

One of the models is QMOOD (Senivongse & Puapolthep, 2015). As Figure 2.11 illustrates, the quality attributes are getting more concrete from the top to the bottom of the model. The second layer covers the design phase, and the third and fourth levels cover the implementation artefacts. QMOOD covers more than maintainability; for example, it includes effectiveness which is beyond the scope of this study.

As Figure 2.11 shows, the maintainability criteria get more concrete going from top to bottom. Level 1 is dedicated to maintainability; the medium level is related to medium-

quality attributes such as analysability, readability, changeability, stability, and testability. Level 2 hosts some more tangible service properties such as coupling and cohesion; and level 3 assesses the properties in level 2; and level 4 focuses on implementation and service components such as service interface, messages, and service interactions.

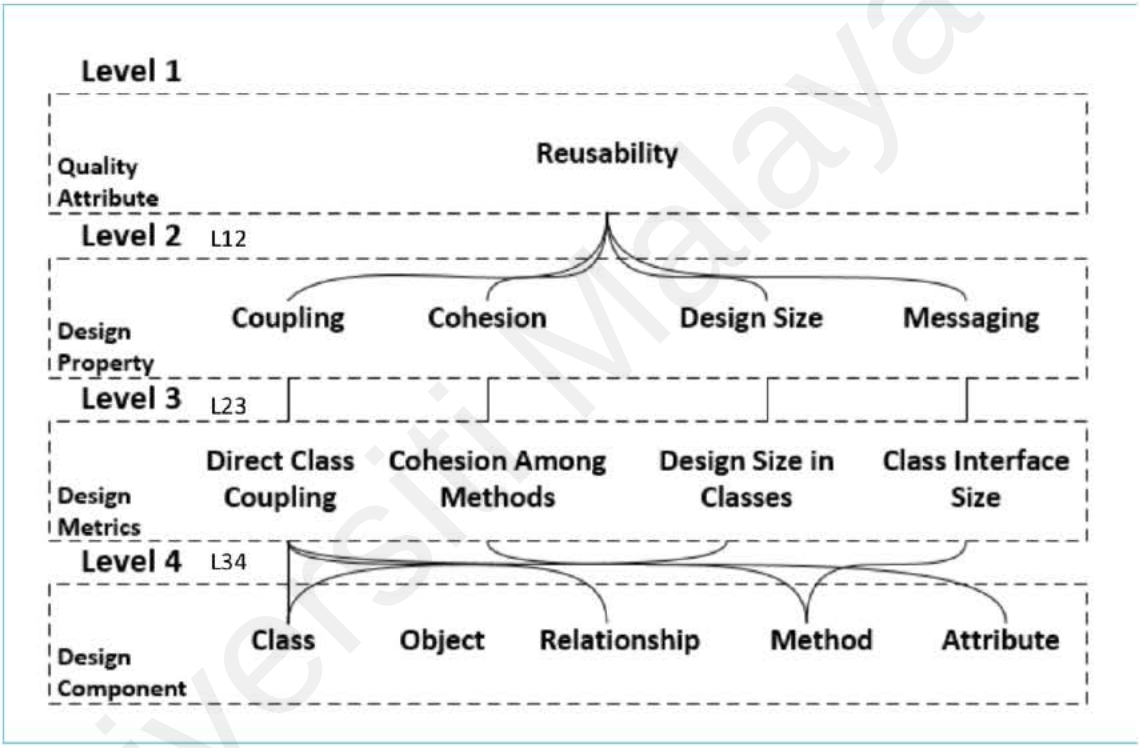

Figure 2.11 A maintainability assessment model for service-oriented systems QMOOD (Senivongse & Puapolthep, 2015)

As the main concern of this research is to improve the coupling between the microservices as the main building blocks of PRA systems, a quality model for the service-based systems is needed for evaluating the improvement achieved in this research. The quality should come with quantitative and descriptive attributes which facilitate the comparison between the systems. Hence, this researcher proposed the adoption of the Maintainability Model for Services (MM4S) (Mitchell & Mancoridis, 2006) to serve as a practical approach. As Figure 2.12 shows, this model is a hierarchical structure.

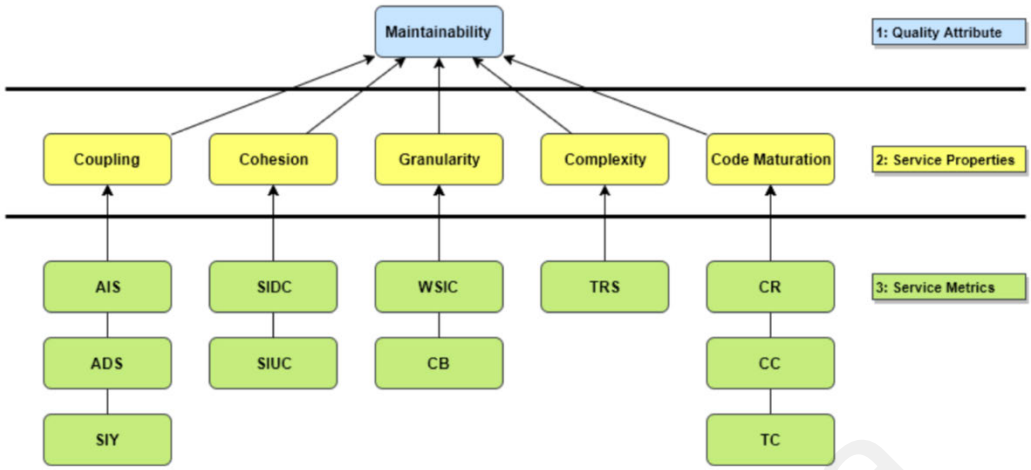

Figure 2.12 Maintainability Model for Services (Mitchell & Mancoridis, 2006)

MicroPRA relies on MM4S as it is simple and introduces quality attributes as concrete service properties. Comparing the hierarchical layers in MM4S introduced in Figure 2.12, with those of QMOOD (Figure 2.11) makes this fact clearer.

As mentioned above, the MM4S only covers maintainability. The second layer involves the service properties of coupling, cohesion, granularity, and code maturation. There are additional similarities between the two models, as can be seen in Figure 2.12 which shows the service properties used in QMOOD model. The strength of the service properties is that, based on the quality attribute, the weight could range from 0.5 to 1 if it has a positive impact on the system; or otherwise, the weight could be -0.5 to -1. This quality assessment technique has also been used in MM4S. While QOOMD was proposed for the object-oriented systems, MM4S has more tangible metrics for microservice-based systems.

The suggested service properties for MM4S are elucidated below.

1- The absolute importance of the service (AIS): This parameter is quantified based on the usage of the services. This property is aligned with the recommendations for QOOMD (Figure 2.11), i.e. average number of directly connected services. QOOMD differentiates between consumers and service providers who consume a service. MM4S does not discuss this much detail.
2- The absolute dependence of the service (ADS): This refers to the number of services that a service *S* depends on. It means that the service *S* calls for at least one function. This property is aligned with the "average number of directly connected services" parameter in Figure 2.113 for QOOMD as well.
3- Service interdependence in the system (SIY): This denotes the number of bidirectional dependencies, i.e. the number of times that service *S1* calls *S2*. In this case, function calls from *S2* to *S1* should be avoided. Therefore, the ideal state for this relation is no call and 0 is preferred.

The comparison of MM4S and QOOMD shows MM4S is simpler and more relevant to quality measurement for microservice-based systems.

SERVICE SYSTEM METRICS

Service System Metric	Definition	
Average number of directly connected services	$\frac{NDPS + NDCS}{SSNS}$	*NDPS* = Number of directly connected producer services in the system *NDCS* = Number of directly connected consumer services in the system *SSNS* = System size in number of services *TMU* = Total number of messages used in the system *NSO* = Number of Synchronous operations in the system *NAO* = Number of Asynchronous operations in the system *NFPO* = Number of fine-grained parameter operations in the system *NINS* = Number of inadequately named services in the system *NINO* = Number of inadequately named operations in the system
Inverse of average number of used message	$\frac{SSNS}{TMU}$	
Number of operations	$NSO + NAO * 1.5$	
Number of services	$SSNS$	
Squared average number of operations to squared average number of messages	$\frac{\left(\frac{NAO + NSO}{SSNS}\right)^2}{\left(\frac{TMU}{SSNS}\right)^2}$	
Coarse-grained parameter ratio	$\frac{NSO + NAO - NFPO}{NSO + NAO}$	
Adequately named service and operation ratio	$\left(\frac{SSNS - NINS}{SSNS * 2}\right) + \left(\frac{NSO + NAO - NINO}{(NSO + NAO) * 2}\right)$	

Figure 2.13 Service Properties for QOOMD

The metrics proposed for MM4S are concrete, and it is applicable to microservice-based systems as well. Maintainability is the top-level quality attribute in the first layer. The second layer depicts the service properties. Coupling was one of the most significant properties considered in this study. The other service properties included cohesion, granularity, complexity, and code maturation, but only the metrics for coupling will be focused in this study. The last layer shows the service metrics. These are numerical

calculations based on the product itself. Table 2.4 shows some measurable metrics in the maintainability model for microservice-based systems.

Table 2.4: Measurable Quality Attributes Based on MM4S

Metrics	Meaning	Description
AIS	The absolute importance of the service	The number of clients that invoke at least one method of the service
ADS	The absolute dependence of the service	The number of services that current service depends on
SIY	Service interdependence in the system	Number of bidirectional dependencies

2.5.5 Coupling in Microservices

Coupling refers to the extent to which a module in the system depends on another module. Coupling and cohesion are the most essential factors for the quality of modularisation in the software industry (Tempero & Ralph, 2018). A bounded context and clearly defined business boundaries are vital for making a microservice business-oriented and loosen its dependencies.

With modularity and layering, tight coupling between technology and domain-specific elements is going to disappears, even in monolithic applications. As mentioned before, there is a significant relationship between a microservice and coupling. Microservice aims to lose coupling between the system modules. Microservices are independently deployable and much more decoupled than monolithic services. Each microservice only concentrates on one business aspect, domain area, or business capacity.

The lower the coupling, the greater cohesion between microservices. (Nadareishvili, Mitra, McLarty, & Amundsen, 2016).

2.5.6 Metrics for Measuring Coupling

Some studies on modularisation and componentisation assessment techniques are present in the literature. The majority of the works are based on object-oriented systems, and the evaluations were done using structural information and dynamic information. Interpreting the structural information involved a static code analysis based on the number of calls between classes, inheritance, and variable access. The runtime executions specified some dynamic information (Russo & Oliveto, 2016).

Based on MM4S, which was discussed in the previous section, the coupling is a service property that falls under maintainability. This provisioning is useful when it comes to the assessment of the decoupling improvement in micro-service architecture. Apart from the categorisation, the service metrics described by MM4S were used as quantitative metrics in this study.

Descriptive metrics form another group of metrics, the most famous and classic of which is the Bunch tool. This tool is a kind of abstraction tool that provides suggestions for clustering a domain. Clustering a domain or, in other words, model decomposition is a vital activity that has been previously discussed in Section 2.5 in the context of Viscovery system. As mentioned before, the authors divided the domain into four different groups. In the other architectures that were discussed before, including PaloPro and even older module-based systems as well, this kind of division can be observed. The Bunch tool suggests drawing a Module Dependency Graph (MDG) for the clustering or model decomposition purpose. The graph would consist of clusters, with each cluster including some edges and nodes. High cohesion is observable based on the number of the edges among the nodes in one cluster, and coupling can be measured using the

dependencies of different clusters. Fitness functions can also be used in this method to evaluate the quality of graph partitions. By using a heuristic search space algorithm, this method provides optimised cluster decomposition (Russo & Oliveto, 2016).

One of the strengths of the Bunch tool is that it introduces an interface and API to integrate with other tools. Further, it is possible to introduce third-party optimisation algorithms. Figure 2.14 shows the extendibility feature of this tool. As a result, this is a kind of framework rather than an application.

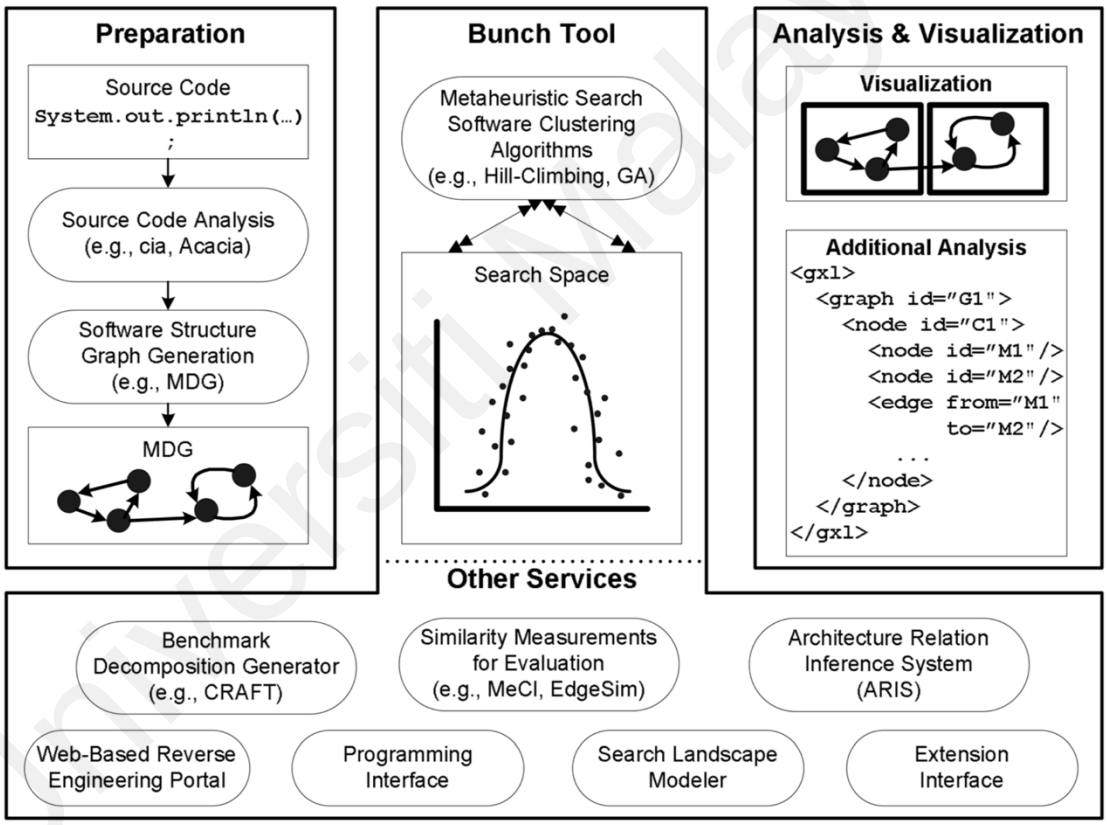

Figure 2.14: Extendibility of Bunch Tool as a framework

Having a proper and effective way of decomposing microservices is a crucial part of microservice-based SDLC, and it is important to define the boundaries such that they preserve the modules. There is a graph-based model called CoCoMe (Tyszberowicz et

al., 2018) that can be used for this purpose. CoCoMe aims to cluster microservices to achieve low coupling, which is relevant to this study. The following section discusses ways for microservice visualization methods to qualitatively assess the coupling.

2.5.7 Coupling in Microservice-Based Architecture

The main aim of microservice architecture is flexibility for change, and it is directly related to having loosely coupled services. By decomposing a domain into functions and specifying functions related to each other, functionally related features can be specified using a cluster (Tyszberowicz et al., 2018). Each cluster is a candidate for a microservice or a group of related microservices.

2.5.8 Design Patterns for Microservice-Based Systems

In microservices, there are some patterns to follow and implement. Some of them are classical patterns inherited from the ancestor architectures, While some of them belong to the microservices ecosystem. As previously mentioned in Section 2.5, one of the main issues related to the microservices is the bounded context. The bounded context has a direct relationship with the older pattern separation of concerns. The pattern gathers together the features that are most likely to change together for the same reason and separates the parts that may change for different reasons (Killalea, 2016). If a change going to affects microservices in different clusters, something is wrong in the design (Mayer & Weinreich, 2018). Therefore, designing the communication occurring between microservices is essential.

There is also an API gateway pattern for the case when there are multiple microservices. While thick provisioning may be required for these services and also different devices may need different formats, the API Gateway will take care of this by sitting in front of a group of microservices. The Viscovery sentiment analysis platform discussed in Section 2.5.2 has implemented this API gateway (Espinoza et al., 2018b).

However, it is mostly an API aggregator and does not perform load balancing and routing tasks alone.

2.6 Summary

This chapter first introduced the domain for the sentiment analysis and review analyses. The existing architectures for PRA systems were discussed with the aim to propose an improved architecture. The microservice architecture was then introduced to highlight its usefulness in PRA systems. Coupling was explained as a software quality attribute that assesses the proposed architecture. Quality attribute models were compared, and the maintainability model was highlighted, as it is suitable for microservice-based systems. A directed graph was introduced to visualize the microservice clusters aligned with business sectors.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

This chapter presents the overall method followed in this research. The study was initiated by a review of existing works, as presented in the previous chapter. The architectures of existing PRA systems, which were extracted from recent works, were also explored. Following this, the problem statement, scope, research questions, and objectives were elaborated further. After that, an improved architecture was proposed, and the advantages and outcomes of the proposed architecture were mapped along with the research objectives. Then, a prototype PRA system was implemented, and, finally, the metrics for coupling and cohesion were used as the quality measurement criteria. Figure 3.1 depicts the research methodology.

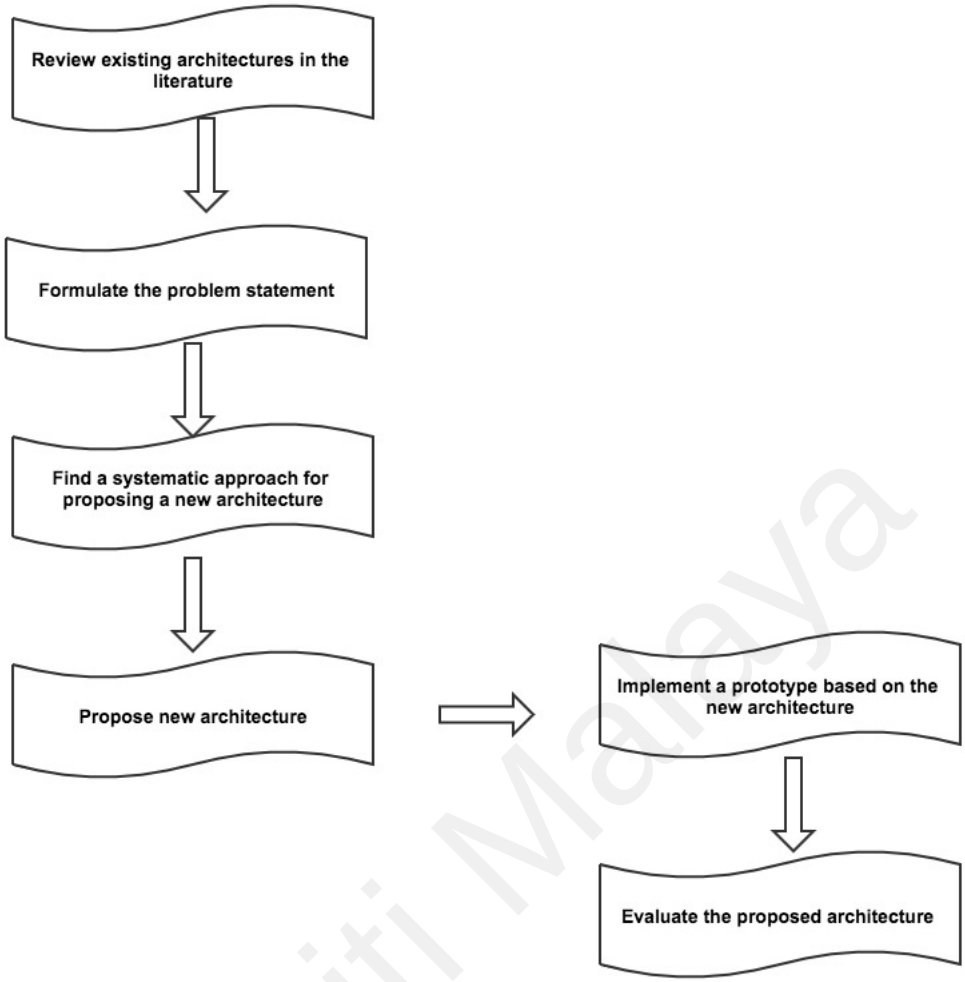

Figure 3.1: Research Methodology

3.2 Review Existing Architectures in the Literature

For conducting this dissertation, a systematic search was performed along with ad-hoc searches. Due to the diversity of the subjects, topics such as opinion mining, review analysis, platform, architecture, microservice architecture, coupling, cohesion, and software quality metrics were used for the searches. Finding the relevant literature demanded outstanding effort and time. The main focus was looking for architectures in the existing literature to find a gap. For this purpose, architecture and platform were the major search criteria always in the literature exploration.

3.2.1 Selection of Documents and highlight the main architecture

Almost all documents for the literature review were selected from journal and conference articles. Generally, online databases such as Science Direct, IEEExplore, Web of Science, and Scopus were searched into. Most of the papers on the subject of PRA system platforms and architectures within a specified area were selected for review to assess fitness to the scope of this study. The architectures presented explicitly or implicitly in the articles were extracted as the main focal point for the analysis. Books and articles were also referred to for extracting fundamental concepts such as stream processing, microservice architecture and software quality metrics.

3.2.2 Selection of Search Queries

Noticeable query results were obtained using *sentiment analysis*, *opinion mining*, and *customer review analysis* as the search query parameters. The search was refined further with the terms *platform* and *architecture*, and there were zero results when "microservice architecture" was added to the search queries. Therefore, most of the documents were selected by using the first set of search items, i.e. sentiment analysis, opinion mining, and customer review analysis.

Based on the literature reviewed, it is clear that architecture and framework were not included among the keywords for most of the studies. To meet the objectives of this study, microservice architecture, architecting microservices, componentisation, modularisation, coupling, and cohesion metrics in software architecture were independently used to find microservice-related references.

3.3 Formulation of the Problem Statement

Even though many of the query results were about solving non architectural related problems, accurate exploration on the search contents led to some explicitly documented or undocumented PRA system architectures. Based on the initial works, the main

components of the architectures and relationships between the components could be identified.

Studying the selected documents from the literature search led to a compilation of the chronology of existing architectures and platforms. The platforms were compared, and their architectures extracted to help identify the research gaps and verify the problem statement.

3.4 Find A Systematic Approach for Proposing a New Architecture

Base on the existing literature it was not clear enough how the module decomposition has been done, how the final architectural component has been mapped into the decomposed modules. This research aimed to find a systematic approach to mapping decomposed modules into architectural components. The core modules for the PRA system extracted from the existing literature. This mapping helps for a better comparsion between new architecture and exiting ones.

After identifying the modules the main problem how to map and convert the modules into microservices as the main bulding blocks for the proposed architecture. For this purpose, the study explored within the existing techniques for microservis decomposition.

Common Component Modeling Example (CoCoME) model is identified as a result for the research (Tyszberowicz et al., 2018). Based on the CoCoMe model, by relying on the use case specifications, the researcher specified certain actions and state variables. The relationships between the action and state variables were used as the input for splitting the domain into subsystems or modules as well as discovering microservices candidates. Considering the microservices clusters and the use of a directed graph to represent them, as previously discussed in section 2.5.6, this criterion is important for the third objective of this research, which is about moving towards a weakly coupled system.

So, the CoCoMe model was the preferred choice for this story to align the CoCoMe model with the directed graphs (Mitchell & Mancoridis, 2006) for measuring the quality.

Figure 3.2 depicts the concrete steps taking in this study to decompose the sample domain into microservices and come up with a new architecture. As mentioned before, the idea for the steps involved in this study, came from an existing work on the CoCoME model (Tyszberowicz et al., 2018). This model has already been introduced in Section 2.5.6. The motivation for choosing this method was the fact that it aligns well with the Bunch model, and there are some structured methods appropriate for decomposing microservices. The steps mentioned above will be further elaborated in the subsequent sections.

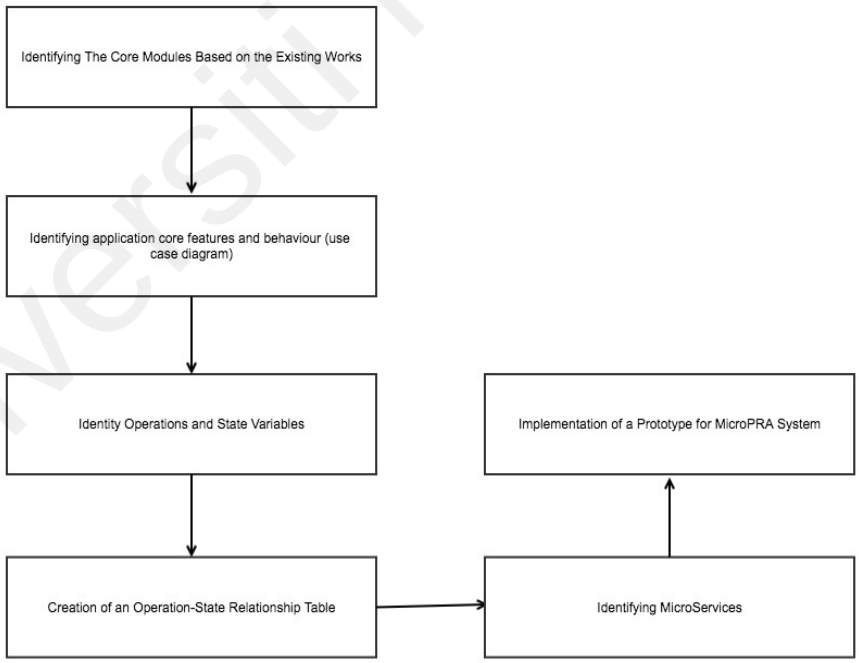

Figure 3.2 Concrete steps for MicroPRA development

3.5 Proposal for a New Architecture and MicroPRA development

The main objective of this research was to propose a new and an improved architecture for PRA systems in order to reduce coupling. First, the general features for the existing

PRA system were analysed based on the literature. The main modules and building blocks of PRA systems were identified by analysing the requirements of PRA systems. The next step was to decompose the modules into reusable microservices. Finally, the complete microservice architecture with peripheral components and a message broker provided the outcome for filling the gap in the existing review, which will be discussed further in later sections.

3.5.1 Identifying The Core Modules Based on the Existing Works

As mentioned before by studying the existing literature, the core modules for a general PRA system is understandable. Therefore, as a first step the identifying modules were significant.

As indentified in the Section 2.3 while exploring the architectures in the existing literature, the PRA subdomain based on the work of Petz et al. (2014b) was decomposed into the modules as shown in Figure 3.3. The descriptions for each module are given below. This study decided to pick this decomposition as it is a quite good candidate and it is somehow similar to the other relevant works. On the other hand, it is a suitable choice for applying architectural improvements and prototype it as proof of the concept for the proposed architecture.

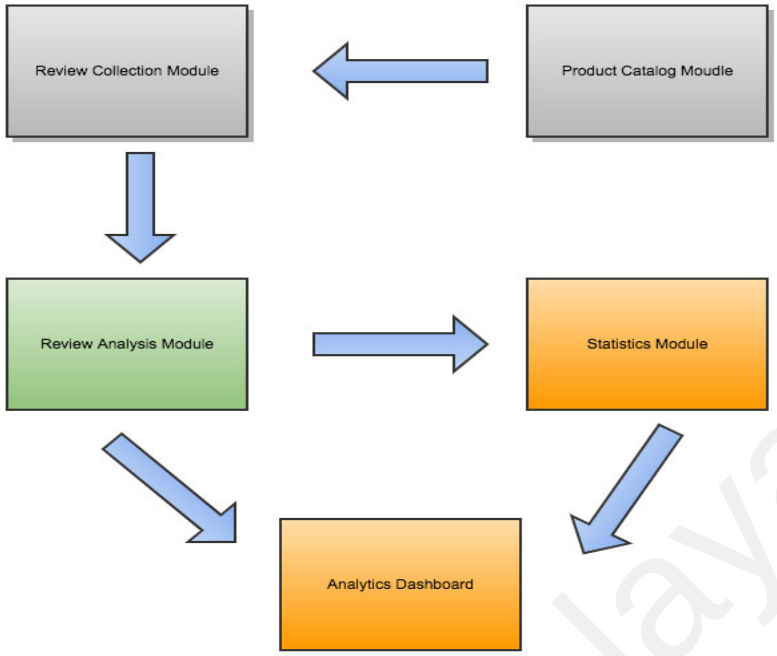

Figure 3.3: New module decomposition aligned with (Petz et al., 2014)

The describtion for the modules are as follows:

1- Product Catalogue Module

This module is needed to manipulate the product and service information along with channels that the PRA system will use to gather online reviews.

2- Review Collection Module

For manipulating reviews, the review collection module is in charge of hosting all the review collection microservices that involve different e-commerce channels and different formats. This microservice plays a major role in the review text supply for the PRA system.

3- Review Analysis Module

The core part of a PRA system is analysis, including review usefulness, entity recognition, context identification, POS tagging, and sentiment analysis. Due to the presence of a variety of algorithms, this study focused on POS tagging and sentiment analysis only. Nevertheless, the efficiency of the algorithms is not the main problem addressed in this dissertation.

4- Statistics Module

To visualise the analytical results, more processing tasks such as aggregation and statistics need to be performed, which this module would be in charge of.

5- Analytics dashboard

Some of the previous researches involved the use of Kibana or DFR Browser (Espinoza et al., 2018b). In this work, for simplicity, analytical bar charts and a word cloud were used.

3.5.2 Identifying application core features and behaviour (use case diagram)

The first step in designing the MicroPRA system is to divide the application's core features into major and critical parts. According to the existing works, as discussed in Section 2.3, generally a PRA system should collect and analyse reviews, and provide analytical reports to visualise the output through front-end applications. The use case diagram in Figure 3.4 provides a reference for PRA system behaviour and operations.

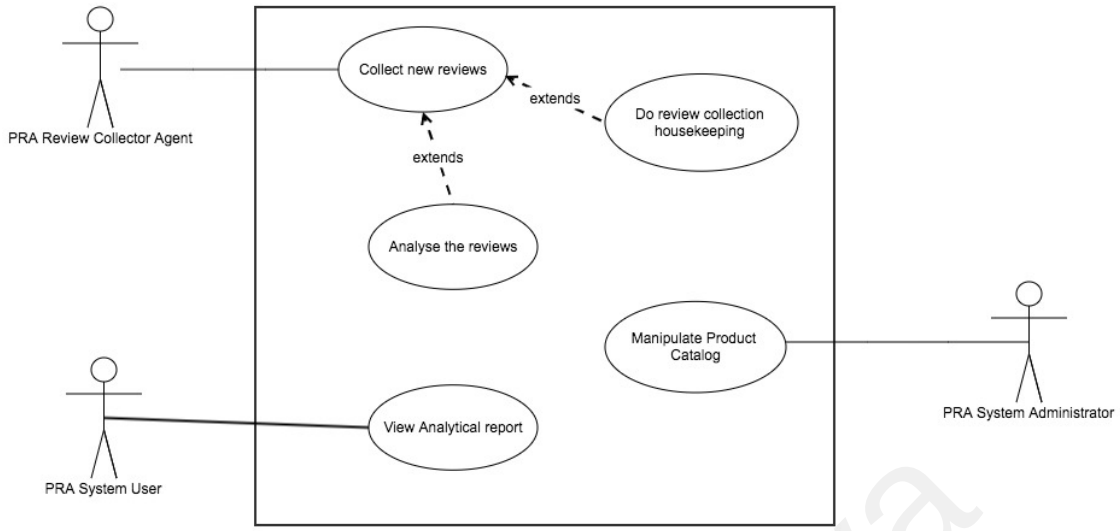

Figure 3.4 Use case diagram for a PRA system

Table 3.1 shows brief use case descriptions. To maintain the simplicity of this work and clarity of discussion, only important use cases were covered.

Use case	**Description**
Manipulate the product catalogue	The administrator creates or updates the product information, including the product's unique ID in each channel, channels for collecting reviews (Amazon, Twitter,etc.), and the endpoints for collecting reviews.
Collect new reviews	The system should periodically grab reviews from different channels.
Analyse the review	The PRA system should analyze the collected review. Some algorithms, such as POS tagging and sentiment analysis, can be applied and stored in the system.
View the analytical report	PRA users can access the dashboard and see the analytical report, including a bar chart and word cloud.

Table 3.1: The primary use case descriptions

3.5.3 Identity Operations and State Variables

Based on the system functionalities and behaviours described in the previous section, the required modules were identified and decomposed in a conventional PRA system. There is no rigid rule for decomposition. Typically, decomposition is done based on the intuitive experience of the developers, but, as a practice, the business capacity or bounded context can be referenced.

Business capacity is the outcome of the business functionality that generates or provides value to the end-users or clients. As a rule of thumb and inspired by the Domain-Driven-Development (DDD) concepts, one microservice was designed for each area of functionalities.

Decomposition is essential. Bad decomposition may lead to a distributed monolithic application. Also, the communication between services as well as data consistency had to be considered when splitting the functional requirements into microservices. Communication frequency and data transfer volume are essential because of the network latency issue. Network latency leads to the challenge of service availability. Decomposition based on business capability or domain model is one of the approaches for identifying microservices. That is why DDD and microservice architecture are a perfect match for PRA systems. As a simple yet common practice, microservices with close functionalities and respond to the same kinds of changes are placed together.

For the aggregate, in this work, in order to achieve microservice decomposition, the scientific techniques were relied upon, and CoCoME and state-actions tables are utilised for the decomposition.

3.5.4 Creation of an Operation-State Relationship Table

As discussed in the previous section, microservices are heuristically and intuitively identified based on the experience of the designers. (Tyszberowicz et al., 2018) One of the approaches extracted from the CoCoME model was to refine the operations and entities as well as the relationships between them. In practice, a business model consists of a finite set of operations and entities. The operations aligned with the use cases help to decompose the services and specify the boundaries. This semantic approach was described by Tyszberowicz et (2018). The authors compared this semantic approach with the manual design, and then obtained acceptable results. This was the reason for choosing this method for the MicroPRA service decomposition.

Table 3.2 shows the relationship between the operations and states. If the relation is a read or write operation, the number is one; otherwise (for read and write), the number is two. The operations were aligned with the use cases depicted in Figure 3.4. Review and Product Catalogue were the main entities in the MicroPRA core domain.

The collect review operation collects reviews and adds it to the database or storage, so the cardinality for the relation between this operation and entity is 1. On the other hand, the analyse review operation reads a review entity and processes it; the processed one is written in the database. So, the cardinality is 2.

Operation/Entity	Review	Product Catalogue
Collect review	1 (write)	1 (read)
Analysis review	2(read and write)	NA
Show analysis result	1(read)	NA
Manipulate catalogue	NA	1 (read)

Table 3.2: Operation-state relationship for CoCoME Model

Based on the dependencies, visualisation was done in a way such that each service had access to its own state variables, which helped identify the clusters of the dens relationship. Dens relationship refers to the relationship between microservices performing functionally related tasks. This is discussed further in the next section.

3.5.5 Identifying Microservices

It is also important for each service to have access to its state variables. Each microservice covers a specific functionality in a way that there is just one reason to change it. Following this rule decreases the density and relationships between modules or subsystems. Figure 3.5 shows how following the CoCoME model led to the identification and extraction of candidate microservices. First, the variables (states) and verbs were specified based on the use case diagram. Table 3.2: was used for accomplishing this task. Each microservice has its own entity and operation space. If one microservice needed anything from another, it was made possible through an API call. Based on this, Figure 3.5 depicts a graphical representation of the operation/entities table. This graph was later used for service decomposition.

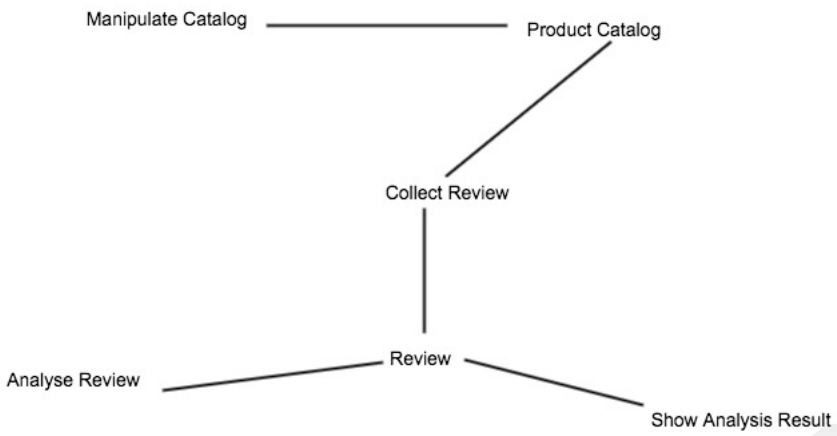

Figure 3.5: Graphical Display for Operation-State Relationship

Based on the dependency graph in Figure 3.5, the candidate microservices were identified as shown in Figure 3.6. Each operation that shared less data with another was a good candidate for a microservice or cluster of microservices. This implies a weaker relationship between microservices that satisfies the low coupling requirement. MicroPRA is a small prototype system, so there is no cluster of microservices. However, it can be assumed that a subdomain such as a product catalogue, in practice, would contain multiple microservices. The relationship between the microservices in a cluster or the internal relationship within a microservices would be dense as they would be large amounts of states or data to share. This relies on high cohesion among the microservices in the cluster.

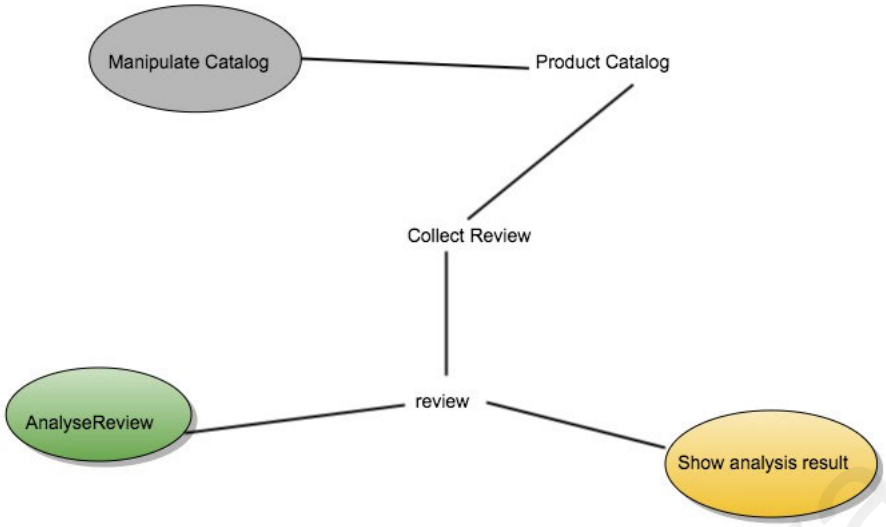

Figure 3.6: Microservices Identified using CoCoME Model

As a result, a meaningful model that guided development of the MicroPRA microservices was obtained. It was important to ensure consistency of the MicroPRA implementation with this model.

Modularisation and the identification of domain clusters are not new tasks in software development. What makes them much more noticeable in microservice development is the tendency towards having much more decoupled services based on the visualization. The visualisation itself was derived from the operation-state table, which was extracted based on the use case. Thus, the requirement was traceable, making CoCoME a useful technique for extracting microservices as well.

3.5.6 Implementation of a Prototype for MicroPRA System

As discussed in the previous sections, improvements were required for MicroPRA system, and certain tools, technologies, and platforms were used to facilitates the achievement of these improvements. Peripheral components such as *microservice registry*, *dedicated API gateway, and message broker*, were the major changes in the implementation. The details will be discussed further in Chapter 5, and the following section covers the important decisions made in detail.

3.6 Important Architectural Decisions

Architectural decisions that helped discriminate MicroPRA from Viscovery played a significant role in this study, as they were aimed to achieve the improvement that was set as the goals for this study. Using stream processing platform (message broker) allowed for reduced coupling between the services. NoSQL database was selected because of its compatibility with unstructured data. API gateway and registry were used in the implementation as supplementary components for microservice architecture.

3.6.1 Stream Processing Platform as a Message Broker

To avoid direct communication between microservices, an event-based system that relies on a stream-processing platform was proposed for MicroPRA. There is no direct service call in the MicroPRA. The event-driven approach decouples the client (service consumer) from the service. Reducing the need for client or service calls was the second outcome of this approach. The stream processing platform employed was the proposed architecture for two reasons. First, it used a publish-subscribe model and decreased the coupling between microservices. The subscriber and the publisher were decoupled from each other, and they did not need to know anything about each other. Second, it is was a stream-processing platform that serves as a middleware, and the PRA system can leverage the scalable characteristic of such middleware. As stated before, one of the issues with PRA systems is the growing volume of the reviews from different channels.

3.6.2 No-SQL Database

A feature of the No-SQL database is the lack of binding to a specific structure (Zablocki, 2015). In the review analysis domain, PRA systems always deal with unstructured data. Even in the review text, the use of emoji and emoticons reveal the sentiment of the review writer. The requirement of grabbing and manipulating this kind of text also leads to the decision to adopt a NoSQL database for the proposed PRA

architecture. The PRA system also does not need as much consistency as SQL-based databases usually do. Thus, the PRA system can enjoy the schema-less feature of the NoSQL database.

3.6.3 Using Supplementary Components for the Microservice Architecture

For achieving scalability and easy provisioning of the microservice, the proposed MicroPRA included a Registry and API Gateway. This decision was inspired by patterns of microservice architecture. Registry and API gateway pattern are recommended pattern for microservice-based systems. 4.4 describes the role for API Gateway and registry for facilitating service provisioning and load balancing between services.

3.7 Evaluation of the Proposed Architecture

As previously discussed in Section 2.5.5, coupling is related to design quality, and it is an essential measurement of the extent to which software components are inter-related. In the following sections, further details about the quality model for coupling are provided. RAs mentioned earlier, measuring all QAs achievements requires a bigger effort and it is out of the scope of this study.

3.7.1 Fine-grained Microservices

A microservices-based system should be composed of fine-grained services. Fine-grained means the size and number of microservices are small enough in order to achieve some quality attributes. There is always a trade-off for the size and number of microservices. (Tyszberowicz et al., 2018). The CoCoME model helps to come up with a more reasonable micro service decomposition and having a conceptual cluster of microservices with less dependency.

3.7.2 Assumptions Made for Viscovery for Comparison with MicorPRA

The necessary details for performing a comprehensive benchmarking is not available for Viscovery. The authors of Viscovery (Espinoza et al., 2018b) did not mention about the microservice decomposition technique they used. On the other hand, CoCoME was utilised for MicroPRA to reach the stage that the candidate microservices could be extracted as a design element. With regard to microservice decomposition, assuming that Viscovery's microservices were aligned with microservice decomposition using CoCoME, MicroPRA's microservices are comparable to that of Viscovery, as CoCoME is a technique comparable with human design. Chapter 6 states further detail on the comparison between the two systems.

3.7.3 Descriptive and Quantitative Metrics

As quoted by Tempero & Ralph (2018), coupling is defined by Yourdon and Constantine as:

> Coupling as an abstract concept—the degree of interdependence between modules—may be operationalized as the probability that in coding, debugging, or modifying one module, a programmer will have to take into account something about another module. (Tempero & Ralph, 2018, p. 215)

As a conceptual rule, the more dependent a component is from the others, the more tightly coupled the components are, making it difficult to reuse and change them. Most literature reviewed concern coupling metrics for object-oriented or structured programs, and they are dealing with dependencies between classes and class hierarchies (Tempero & Ralph, 2018). In the context of this study, microservices are more granular, for which there are no well-defined metrics yet. That is why a descriptive method was also chosen as the quality assurance tool for this study.

Apart from using the dependency graph as a descriptive method, another quantitative method selected for assuring the quality was MM4S. This method was chosen because it concerns method calls, and due to the layering architecture used for implementing microservices, it is a suitable metric for measuring internal dependencies between the microservices. MM4S was devised specially for microservice-based systems. It was inspired by the previous quality model used for object-oriented and service-oriented systems. (Senivongse & Puapolthep, 2015)

3.8 Summary

The steps involved in forming the proposal for the MicroPRA as well as the reasoning behind the methods for extracting the microservices as a significant step in the design have been discussed in this chapter. The important decisions taken for the MicroPRA design and rationale for the quality assessment model were described as well. The detailed architecture of MicroPRA will be discussed in the next chapter.

CHAPTER 4: THE PRPOSED ARCHITECTURE (MICROPRA)

4.1 New Architecture Desicions

According to the approach for decomposing a PRA system into microservices as discussed in Section 3.5, the proposed microservice architecture is shown in Figure 4.1. The microservice registry, streaming processing software platform (or message broker in this context), separate databases, and a gateway with newer functionalities are architectural decisions proposed for MicroPRA which are different from the existing ones, like Viscovery.

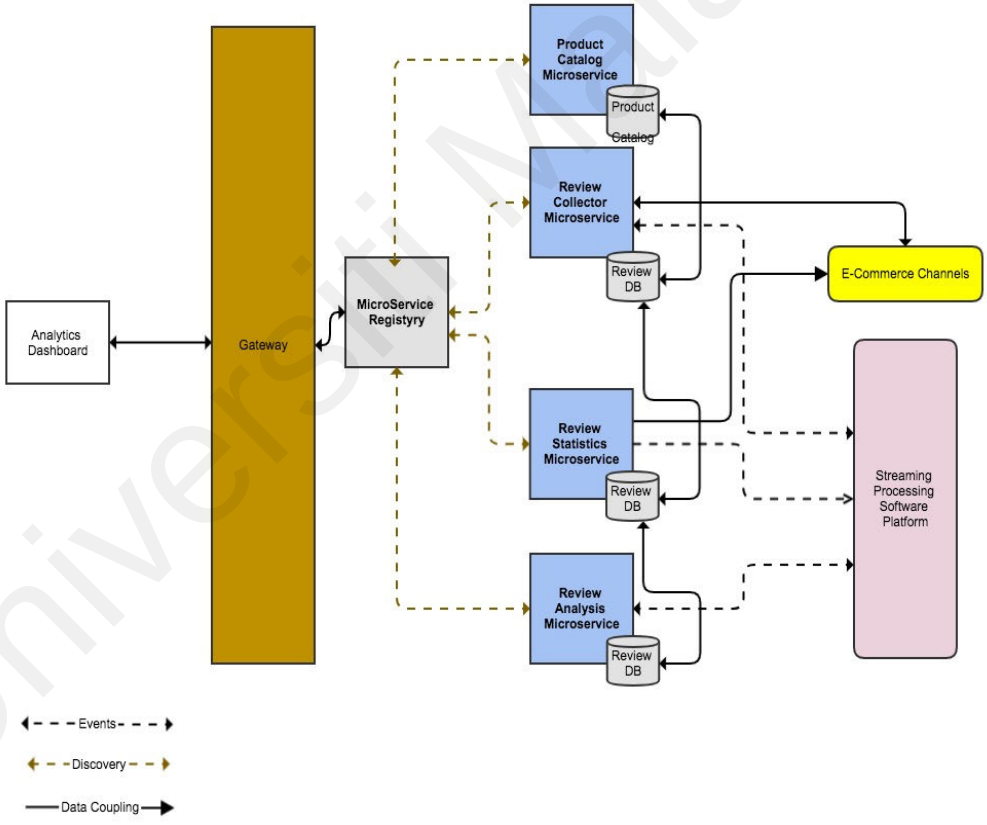

Figure 4.1 Conceptual View for the Proposed MicroPRA Architecture

4.2 Comparsion between MicroPRA and Viscovery

Figure 4.1 shows the proposed architecture adapted from Viscovery. MicroPRA architecture has an auto-discovery feature for the web services, and the API gateway role is aligned with that of Viscovery. It is worth noting that the discovery feature sometimes can be delegated to the deployment infrastructure, but in MicroPRA, each service needs to undergo a configuration for auto-discovery.

The following results could be obtained by comparing the architecture of MicroPRA in Figure 4.1 with the architecture of Viscovery in Figure 2.8 :

1- A gateway was introduced in MicroPRA for routing and load balancing.
2- The service registry provisions the MicroPRA services, but the component does not exist in Viscovery.
3- Communications in MicroPRA are via a message broker and through the events. Viscovery does not have message broker and all communications between the services happens through direct call.
4- Viscovery Novaviz API gateway is an entry point for the utility services for review processing, but MicroPRA API gateway is the only entry point for the entire system.

Table 4.1presents comparative view between Viscovery and MicroPRA in terms of the main architectural componsts.

Table 4.1 Architectral Comparsion between MicroPRA and Vsicovery

Component	Component Type	MicroPRA	Viscovery
Injestor	Main functional component	√	√
PreProcessor	Main functional component	Not available*	√
Processor	Main functional component	√	√
Indexer	Main functional component	Not available*	√
Event publisher	Non functional	√	Not available
Service Registry	Complimentary Component for microservice architecture	√	Not available
Auto discovery	Non functional requirement for microservice architecture	√	Not available
API Gateway	Complimentary Component for microservice architecture	√	Not available**
Frontend ***Dashboard	Main functional component	√	√

*Implementing all the equivalent components same as indexer and preprocessor are out of the scope of the MicroPRA as it requires much more effort. Leaving out this functional component does not have any effect on the evaluation of the improvement.

** As mentioned before, what Viscovery calls as API gateway (Novaviz) is not what MicroPRA meant by API gateway. API gateway in MicroPRA works as a complementary component required for microservice architecture. It does routing between microservices. Novaviz is kind of integrated API for NLP and other required utilities for review analysis.

*** The purpose of front-end dashboard is different, as Vsicovery is meant for improving topic browsing but MicroPRA is meant for improving the architecture.

4.3 Decompose a PRA System into MicroServices

Using a systematic approach (CoCoME) as described in Section 3.3.4, microservices required for building MicroPRA were identified. The microservices in the proposed architecture are discussed in the following sub-sections.

4.3.1 Product Catalog Microservice

This microservice manipulates the product or service information, including the channels that the PRA system can go through for collecting reviews. Different e-ommerce channels have different identifiers for each product, so this microservice needs to identify those as well.

4.3.2 Review Collector Microservice

The collector microservice collects reviews from different e-commerce channels. Each channel has a different structure for the reviews, and their communication channels are also different. Some of them do have a well-defined API for exposing product reviews to external parties, but others do not. Hence, there is a need to hire an HTML scraper or web crawler for the latter.

4.3.3 Review Statistics Microservice

This microservice is keeping track of statistics and it is a kind of housekeeping analytical service that decides how often the reviews should be collected by the collector microservice. An existing architecture, Paolo Pro, includes a feature of the prioritisation of resource such that the platform would decide which resource would be prioritised for collecting the reviews (Tsirakis et al., 2015). This microservice can identify the products getting noticed for having more reviews and on which channels. The proposed housekeeper microservice performs a similar functionality, including prioritising products to review.

4.3.4 Review Analysis Microservice

Review analysis is the core business of any PRA system. Most of the e-commerce channels provide user rankings, but in the majority of the cases, the user rank does not indicate the customer's feelings about the product. Therefore, quantifying sentiments as negative, positive, or neutral helps review analysts to verify the helpfulness of a review and the correctness of a user ranking (Cambria et al., 2017). Helpfulness, from the customer's perspective, refers to the extent to which a product meets their expectations, which helps suppliers identify defects and improve the product.

In this work, the Stanford NLP library was used for the purpose of sentiment analysis. It includes a variety of algorithms, but only sentiment and POS tagging features were engaged in the proposed sentiment microservice. This will be elaborated further in Section 5.2.6.

4.4 Supplementary Components

Despite the advantages of microservice architecture, there are also some pitfalls. For instance, managing and monitoring the microservices is hard. Thus, some supplementary components were introduced.

The microservice registry was one of the components, and it facilitates keeping the dynamic locating of microservices and helps improve scalability. The purpose of a service registry is to obtain information about which and where a service is running. To provide more scalability in practice, most high-volume PRA systems would have more than one instance running from one microservice. The registry keeps track of the existing microservice instances and dynamically locates them (Balalaie et al., 2018). The idea is simple; each service needs to register its network location with a service registry. When a new request comes in, service discovery queries the service registry to get a list of available instances for the related service.

Another supplementary component is the microservice API Gateway. The API gateway routes the requests to the service and composes the responses. It also performs authentication and rate-limiting tasks. Rate-limiting limits the requests for either a specific client or all clients. When there are many microservice instances, load needs to be balanced between instances, since one microservice may serve other microservices as well. That is why a load balancer was a part of the gateway too.

Some architectures limit the API gateway component to routing and composition only. In such cases, authentication and the other functions are delegated to a new service, which is usually called an edge service. One advantage of this is the better separation of concerns. Each request would first be received by the edge service and, after processing, be routed to the API gateway. The major drawback, however, is network latency, though. In the proposed PRA architecture, all the functionalities are concentrated in an API gateway.

4.5 The Outcomes of the New Architecture

In this architecture, a simple dashboard has been exposed as a porotype to consume the REST API that microservices present as well as visualise the output for the users.

As explained earlier in Section 2.3, Viscovery is the latest PRA system based on microservice architecture. By comparing MicroPRA with Viscovery, some findings have been highlighted to address research questions 1 and 2 mentioned in Section 1.3. However, since there is no access to Viscovery's implementation documents, this comparison was only based on assumptions made from the information available in the literature

In Section 2.5.32.5.4, the MM4S model has been introduced as the selected quality model in this study. One of the focus points for the improvement of the system is

coupling. Coupling is an abstract quality model and the underlying service property for maintainability. According to MM4S, the service metrics for coupling are AIS, ADS, and SIY. Based on the proposed architecture and the relations between the services, the service property metrics were calculated. Table 4.2 shows the metrics for coupling based on the MM4S.

1- AIS:

Since MicroPRA utilises an event-based system, there is no direct communication between services. On the other hand, the communication between services in Viscovery is not countable as there is no such detail to count. Further, based on the article reviewed, each component of Viscovery uses utilities as services, so there are direct calls between components and service. For MicorPRA, the only mentionable consumer is the system dashboard that directly interacts with the API gateway, which is why only the calculated values for AIS were mentioned in this dissertation. Viscovery gets much more weight for this service parameter.

2- ADS:

As indicated by the absolute dependency of services (ADS) shown in Table 4.2, MicroPRA certainly has better value since it uses the event publishing method to decouple services. All the services communicate in an asynchronous manner using event publishing and subscribing methods. Based on the authors' description, no such design decision was made for the Viscovery architecture, and there is a dependency between injector, analyzer and indexer components.

3- SIY:

In terms of independence, MicroPRA's services are independent because of the leveraging of the message broker role in the architecture. This is not true

for Viscovery as there is a coupling between the components, and Viscovery is a component-based system that uses microservices as utilities. For this reason, in this comparison table, SIY value is assumed to be at least 4 for Viscovery. Again, due to the lack of detail about Viscovery's architecture, accurate values were not achievable.

Table 4.2: Service Property Metrics for MicroPRA

Metric	MicroPRA	
	Value	Description
AIS	1	The dashboard portal invokes the API for analytics
ADS	0	Because of event publishing and subscribing, there are no direct calls between services.
SIY	0	There are no direct service calls or dependencies between the services.

Table 4.2 shows that MicroPRA achieves the research objectives that are described in Section 1.4.

4.5.1 Scalability

The scalability of a system is its ability to deal and cope with more stuff with additional resources, including users, requests, data, messages (Brown, 2014) . The choice of technology and protocol can impact meeting scalability as a non-functional requirement.

Using registry as an additional component in the improved architecture would bring more scalability in the case of system expansion. In case there is a need for more than one instance of a microservice, the registry keeps track of the new instances and their availabilities, and the API gateway gets a list of available nodes from the registry. Therefore, in case the volume of requests increases in MicroPRA, it is possible to add multiple instances for the services using the API gateway and registry, and the system does not experience degradation.

Future development would also be much easier as some independent microservices publish the results as events. The independent team can work on different microservices and improve the efficiency of different functionalities, including review collection, analysis, and visualisation. When a microservice asynchronously receives all the information required for the event, the demand for availability will be relieved.

4.6 Summary

In this chapter, the MicoPRA architecture has been explained in detail, and some analytical results were obtained to show the achievements of the proposed architecture of MicroPRA as compared with Viscovery. The next chapter will focus on the technical details for implementing the MicroPRA.

CHAPTER 5: IMPLEMENTATION OF THE NEW ARCHITECTURE

5.1 Introduction

In this chapter, details about the implementation of the MicroPRA are be highlighted to address research question 4 mentioned in Section 1.3, The technology stack used for this purpose will be described further in the subsequent sections.

5.2 Technologies Used

To develop the prototype, the technology stack shown in Table 5.1was used. The source code is open source and hosted on the researcher's GitHub[4].

Language/platform/library	Type	Purpose
Java8	Programming Language	Developing the microservices
Spring Boot 2, Jhipster	Framework	Facilitate the development of microservices
CouchBase	Database	NoSQL Database for handling unstructured text
Apache Kafka	Stream processing middleware	More decoupled microservices
Stanford NLP	Sentiment analysis library	Used in review analysis microservice
HTML, BootStrap, Angular, TypeScript, Word Cloud	Front-end Dashboard	Visualisation

Table 5.1: Technology stack used in the PRA prototype implementation

5.2.1 Apache Kafka

Kafka is a stream processing middleware that can perform processing tasks using multiple pipelines, including aggregation and enrichment, where raw data is consumed

[4] Refer to https://github.com/medimohammadise/review-analysis , https://github.com/medimohammadise/cra-dashboard

from Kafka topics (kafka.apache.org, 2019). With Kafka, the PRA system can publish and subscribe to streams the way a message queuing system does. Kafka stores streames for durability and failover. Kafka can span and run on multiple cluster nodes, which is suitable for scalability.

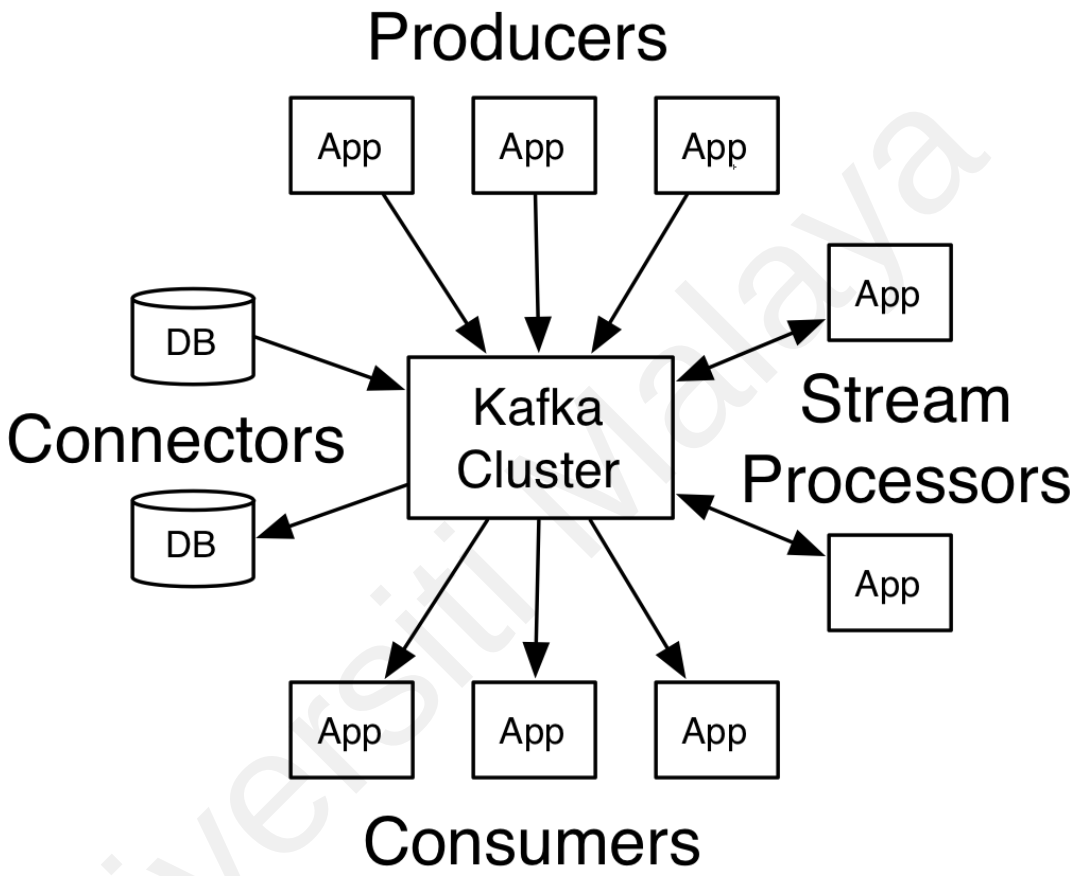

Figure 5.1 Kafka Cluster (kafka.apache.org, 2019)

The purpose of using Kafka in this architecture is to decouple the microservices further. Each microservice just publishes events rather than calling the required functions. Event listeners in the relevant microservices that can grab the events and perform the actions. Kafka is a crucial component in this architecture, as it helps increase the performance by asynchronously running some works.

5.2.2 Jhipster Supplementary Microservice Components

In this work, the existing supplementary components for Jhipster were used to implement the Jhipster Registry and API gateway, as shown in Figure 5.2.

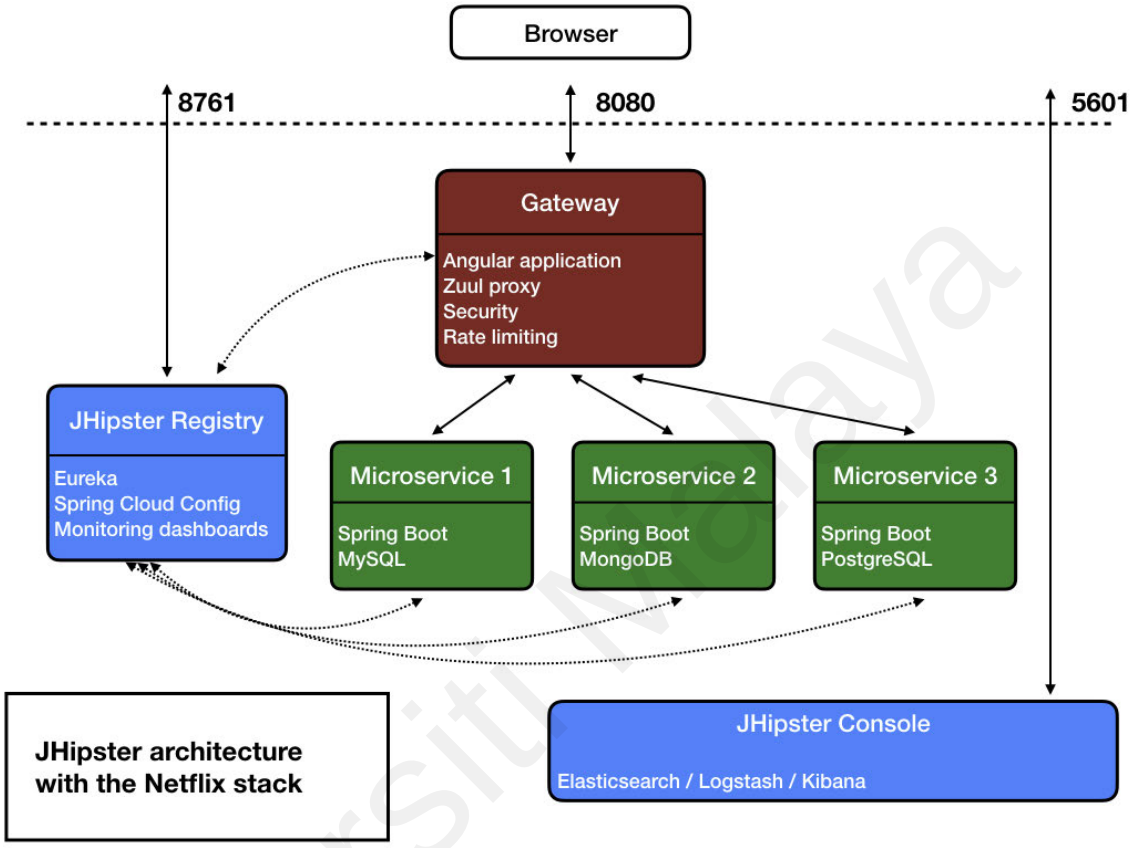

Figure 5.2: Jhipster Registry and API Gateway in the architecture diagram(https://www.jhipster.tech/, 2019)

5.2.3 Jhipster Registry

The Jhipster registry is an open-source Apache2 licensed application that is based on Netflix Eureka. Using Eureka features, the registry can handle routing, loading, and scalability for microservices (https://www.jhipster.tech/, 2019). It also comes with a dashboard for monitoring services and managing applications. Figure 5.2 illustrates all the communications from external consumers, such as Web UI, are possibly using the gateway. The Gateway balances the requests based on the information that it receives

from the registry about available services. These two components help to detect the running services and routing to the available services.

5.2.4 Jhipster API Gateway

Jhipster also generates an API gateway as the front-end for all microservices and provides load balancing and rate-limiting, security, and quality of services and API documentation for all microservices.

5.2.5 Couchbase

Couchbase was the NoSQL database used in this work. It is document-based, and the clustering feature supports scaling out when the amount of review collection and analysis work increases.

5.2.6 Sandford NLP Library

The Stanford NLP library was chosen for sentiment analysis in this system. It is easy to adopt and simple to use as it is annotation based. Stanford NLP is a Java-based toolkit and supports most core NLP processing, including tokenisation. It is also a pipeline-based framework (Manning et al., 2014). Figure 5.3 shows the overall architecture for Stanford NLP. Raw data is fed into the framework, and through each pipeline, the data is getting enriched with annotations. The resultant annotation would contain all required analysed data. In this study, a tokeniser and POS pipeline were used. The tokeniser converts the text into a sequence of tokens, and the POS labels the tokens with its part of speech.

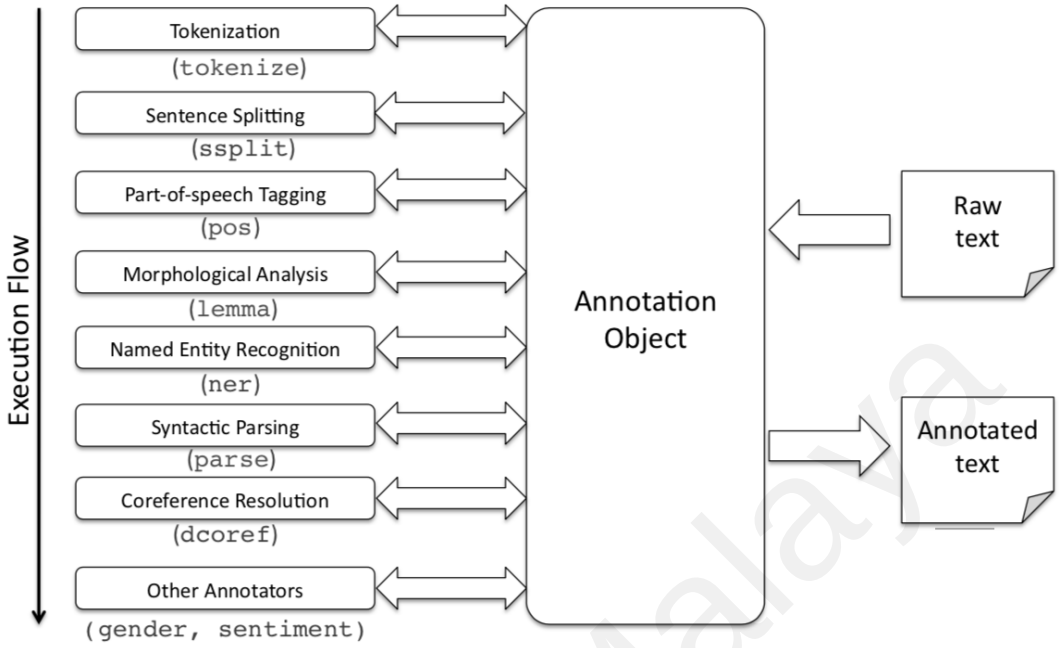

Figure 5.3 Overall architecture of StanfordNLP (Manning et al., 2014)

5.3 How Does the Prototype PRA System Work?

The implemented PRA system is a simple form of event sourcing. In the event sourcing for each domain object, some possible events that change entity state in the event life cycle need to be specified. Each service emits some events in order to indicate the outcome of each operation or consumes events published by the other services. For example, a review needs to be collected, which is an event that the housekeeper microservice emits. As another sample, the review collected is emitted by the review collection service, and the same goes for an analysed review. Table 5.2 provides the list of events and the microservice that triggers each event. Event sourcing helps the microservices to achieve higher decoupling. As another benefit, it helps to maintain the consistency between different databases for microservices.

Publisher Microservice	Event	Subscriber
Review housekeeper	Review(s) ready to collect	Review collection microservice
Review collection microservice	New review(s) collected	Review analysis
Review analysis	Review(s) analysed	Review collection

Table 5.2: Event Publisher and Subscriber Services in the MicroPRA System

By having a message broker, each microservice publishes the messages in specific channels, and the message broker delivers the message to the receiver. Thus, a microservices does not need to directly communicate with another. A microservice could continue its work without the need for blocking and waiting for the response. This idea is depicted in Figure 5.4.

This decision was made for the architecture because minimising the synchronous communications between microservices makes them loosely coupled, and, in turn, improves availability. For instance, when any new review is available for any product, the message broker buffers the message if the review collection microservice is not available. As soon as the review collection microservice is live, the message is delivered to the microservice. The event-based architecture makes the application much more resilient because of this feature (Rihcardson, 2019).

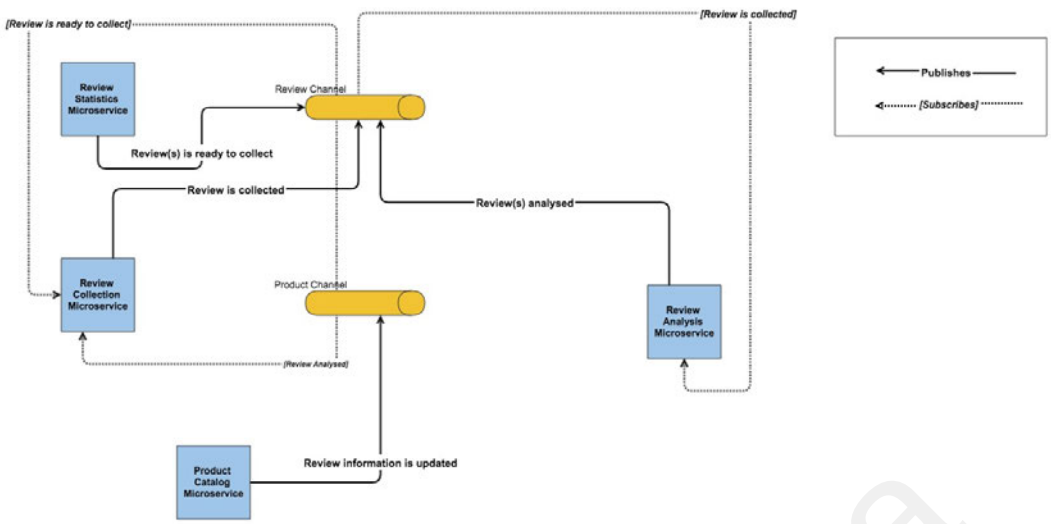

Figure 5.4: Event Channels in MicroPRA Message Broker

It should be noted that, for simplicity, the MicroPRA does not fully utilise event sourcing, as one of the recommendations is to store the event order and recreate the entity state based on the events. The main benefit of this would be a reliable audit log and temporal queries.

This architectural style also provides the capacity for the MicroPRA to use a notification service for the clients. In addition, event publishing facilitates predictive analysis and early notification to the users. For instance, the MicroPRA users would be informed just after receiving any negative review or if the frequency of negative reviews increases or exceeds a predefined limit. The architecture also makes MicroPRA easier to integrate with external or new features by the event-based approach.

5.4 Front-end Dashboard and Analytic Visualization

For visualisation, a simple front-end dashboard was developed. Some external open-source frameworks such as *CoreUI* and *Word Cloud* were employed for providing the

output. For collecting the reviews, two channels were chosen, i.e. Lazada[5] and Amazon. Figure 5.5 shows a sample output of a visualisation component.

The MicroPRA collected reviews for the product "Apple iPhone X, fully unlocked 5.8, 64 GB Space Gray". The channels for the collection were Amazon and Lazada, as mentioned above. The reason for choosing Lazada was to compare the regional attitudes for buying iPhoneX. MicroPRA obtained reviews from Lazada Malaysia. Further, Amazon, being as an international market, was a good choice to refer to what people in other regions thought about iPhoneX. After collecting the reviews, MicroPRA fed them into sentiment analysis microservice. Sentiment analysis processed the text using Stanford NLP and provided the result. Polarity for each review text is obtained. By calculating the average for the polarities, very interesting results could be extracted. Figure 5.5 depicts this information. In the aggregate, more than 60 percent were found to be satisfied with iPhoneX in Malaysia. The review texts analysed were from 2019 in Malaysia. The aggregate percentage was even higher for the UK at the end of 2018. The trend of customers liking the product and being satisfied with it could be measured using these analytics.

[5] Lazada Group is a Southeast Asian e-commerce company founded by Rocket Internet in 2012, and owned by Alibaba Group. In this case for review collection we used Malaysia region from www.lazada.com.my/

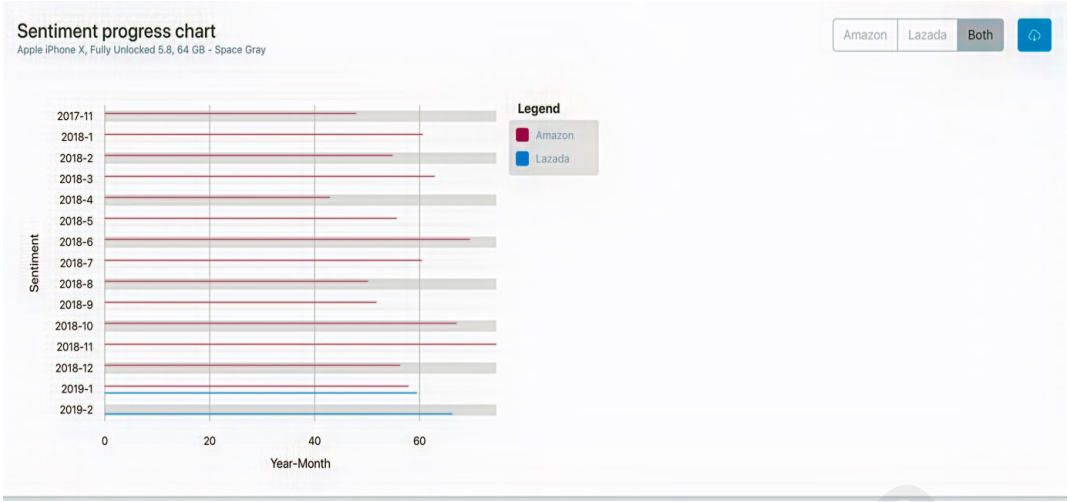

Figure 5.5: Sentiment comparison in Lazada and Amazon for iPhoneX in MicroPRA

For presenting the chart in Figure 5.5, the front-end communicates with the API gateway and invokes the reporting API from the analytics microservice. Lazada turned out to offer iPhoneX starting from 2019 as that's when the product reviews began. The highest recorded satisfaction and positive sentiment were found in October 2018 on Amazon UK. The output could also be visualised through a word cloud, as depicted in Figure 5.6.

Figure 5.6: Word cloud from reviewing text gathered from Lazada and Amazon

5.5 Summary

In this chapter, the implementation of the MicroPRA architecture was explained. The technology stack used was elaborated. The sample outputs for the system were included to show that MicroPRA runs correctly. The reasoning behind the implementation concerns was given in this chapter as well.

CHAPTER 6: EVALUATION OF THE PROPOSED ARCHITECTURE

6.1 Introduction

This chapter describes the evaluation of the proposed architecture, including, decreased coupling and improved maintainability, ease of deployment, and scalability achieved in MicroPRA. As previously mentioned in Section 3.7, two types of software quality assessment models were chosen to evaluate the MicroPRA. The first was qualitative, and the second one was quantitative. The qualitative method was used for descriptively analyse the architecture, and the quantitative method was used to analyse, design, source code, and implementation of the system.

For the qualitative method, a Directed Acyclic Graph (DAG) was used to visualise the dependencies, for better comparison and analysis of the achievements. As discussed in 3.7, the quantitative methods involved the use of MM4S for checking the maintainability of the MicroPRA.

Even though the use of quantitative methods requires details about the implementation of the architecture to be accessible, since no detail information is available for Viscovery, MM4S is used to determine the extent to which the MicroPRA brings improvements.

Qualitative or descriptive parts need evidence and experience along with the concepts to assess achievement. Experience is required to double-check the extent to which the software addresses the requirements concerning the design and architectural decisions.

6.2 Evaluation Result using Qualitative Methods

The MicroPRA architecture has previously been explained in Section3.4, using some architectural diagrams that represent the microservices and relations between them. In this section, as a base for the comparison and better understanding the dependencies between services, a Directed Acyclic Graph (DAG) is used for representation. This

method is a popular one to represent in the modular systems, and microservice architecture facilitates modularity in the context of autonomous services. Generally, microservice architecture tackles complexity by modularising the system. Services are designed based on a business' capability or business context. For each functional area of the PRA domain, a service could be defined.

DAG is the proposed method for tracking the dependencies between services as suggested by Chirotti, Reilly, & Rentz (2018). DAG can be used as a visual reference for tracking and controlling dependencies as well. Another advantage of DAG is the avoidance of cyclic dependencies. Tracking is a passive approach but controlling is an active approach. (Chirotti, Reilly, & Rentz, 2018)

There is a functional alignment between the MicroPRA microservices and some of the components of Vsicovery, As mentioned before, the detail about Viscovery's internal architecture is not completely available. So, relying on some of the assumptions based on the overall findings from the architecture shows that there are still some direct dependencies between the services in Viscovery.

As explained in Section 2.3, certain Viscovery components interact with each other, Data Injector component collects reviews from different sources, such as Twitter and web forums, using a web crawler. The web crawler is a microservice provided by the API gateway.

The Preprocessor component calls microservices such as *stopword-removal, cpas normalisation, symbol/punctuation removal* as well as *dynamic topic models* and *sentiment analysis microservices* using the API gateway. The Indexer components is used for opinion search and retrieval.

By depicting the above scenario using DAG, dependency models of the services were compared in this study. Figure 6.1represents the explained scenarios.

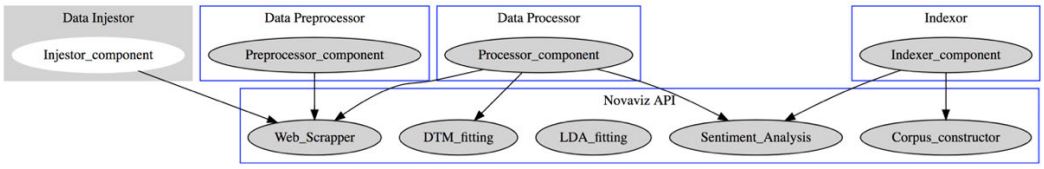

Figure 6.1 Viscovery representation with DAG

Figure 6.1 shows the communications between the Viscovery components and microservices. The representation clearly shows that even though Viscovery utilises Novaviz API, the high coupling is seen in the graphical visualisation. As a result, the Novaviz API gateway also seems to be the central aggregator for utilities that are exposed as microservices. So, the gateway role in Viscovery is different from that of the MicroPRA. In the MicroPRA, the gateway performs routing and load-balancing roles only. As previously mentioned in Section 3.6.3, the Gateway was introduced as a supplementary component in the MicroPRA architecture. In contrast, by modelling the dependency of MicroPRA microservices led to Figure 6.2. As mentioned before, MicroPRA leverages on the advantages of a message broker.(Secion 3.6.1)

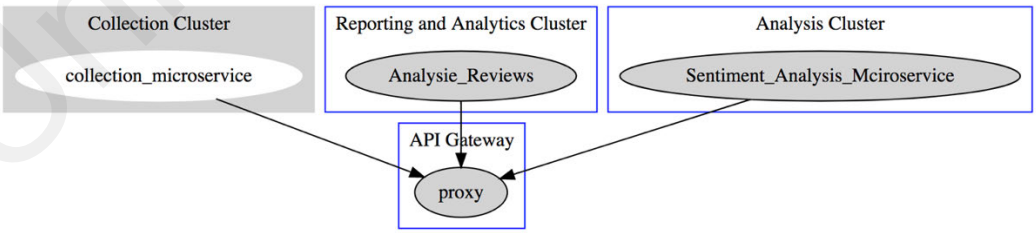

Figure 6.2 Dependency Graph in the Prototyped MicroPRA

Communication between microservices without having a message broker in the architecture makes the microservices highly coupled to each other. Apart from having higher dependencies between microservices, in this kind of architecture, if one service is

not available, the whole system's functionality will get degraded or fail. This issue decreases the system's availability. In the aggregate, degrading service availability could be cascaded through the entire system due to the number of dependencies between services.

On the other hand, using a message broker increases the effort required to maintain the system, as the message broker itself, as a component, requires administration and maintenance. The message broker was used in MicroPRA, as it provides greater decoupling and, in the case of an increasing volume of customer reviews, the message broker provides greater scalability using stream processing features.

Another issue with Viscovery is that, although it uses an API gateway, the authors never mentioned whether this gateway checks for the available services; i.e. it is not clear that whether it uses the functionality of the service registry to understand which service is running and where the service is running. Assuming that Viscovery does not utilise the service registry feature, this will degrade the Viscoverys's availability and reliability as well.

On the other hand, in MicroPRA, as mentioned before, each service has its own database schema. The API gateway or edge services aggregated the API as the front end of a group of microservices, and clients need to talk to the API gateway. Each service can be independently deployed. This fact has been reflected in the DAG graph given in Figure 6.2.

As MicroPRA uses event-driven architecture, there is no direct RPC or API call in the proposed architecture. So, it does not make sense to draw a dependency between the microservice and the event publisher. After each microservice performs its job, it immediately emits an event through the message broker.

Figure 6.2 shows the dependency between the API gateway and the collection, analyse_review and sentiment_analysis microservices. The API gateway is responsible for routing and load balancing between services. There is no dependency between the API gateway and microservices. The API gateway is a bridge to reach each service when a request is received by MicroPRA. All these services get registered in the registry. The autodiscovery feature has been discussed in Section 4.4. As a result, from the DAG graph, a weak dependency between the microservices and registry is assumable.

Microservice's dependency on the message broker can be considered as a kind of a weak dependency as it is asynchronous. Dependencies between microservices and databases are considered as data coupling.

Relying on the events is a significant change toward higher availability, and less coupling. with a message broker, as each service does not need to be aware of the other services and their locations. The service only publishes an event to the channel. Figure 6.3 shows how the dependencies between the services would be without the message broker. Each service would call the adjacent service, and it is hard to maintain this sort of a system. Events change into external system calls. External service calls would improve the coupling metrics (AIS, ADS, SIY) of the MM4S model discussed in Section 2.5.4. That means that there would be greater coupling between the services and the entities in such a way that entities get shared between the services. Next section describes the quantitative results obtained.

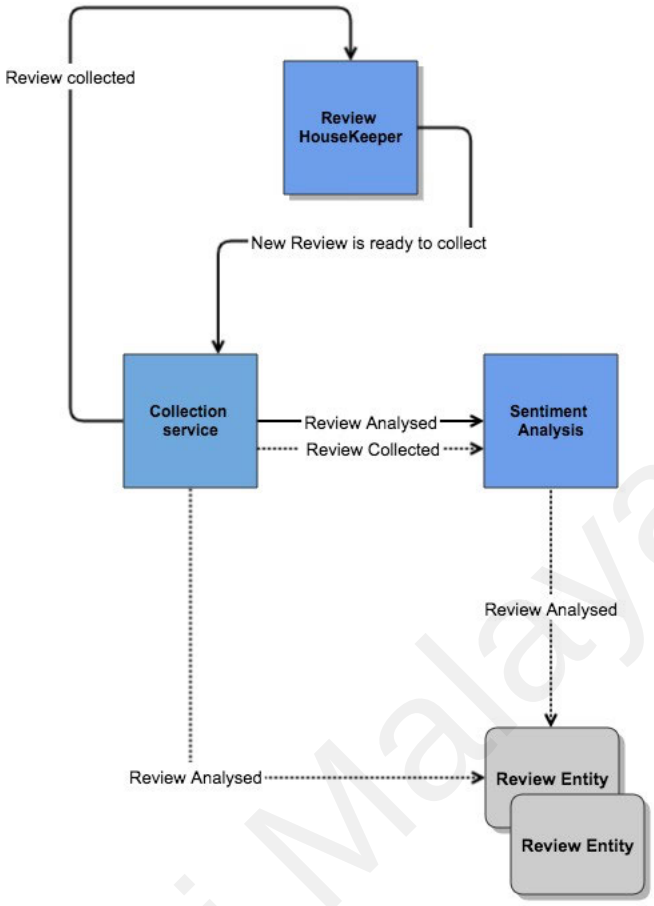

Figure 6.3: Dependencies between the Services in Broker-Less Architecture

To summarise the abovementioned descriptions, Table 6.1 represents the comparisons between MicroPRA and Viscovery.

Table 6.1 Comparing MicroPRA and Viscovery based on qualitative metrics

Metric	**MicroPRA**	**Viscovery**
Scalability	Microservice registry	Does not exist
Maintainability	API Gateway (Load balancer and router)	Exists just for the exposed microservices. It does not perform load balancing.
Low coupling of services	Message broker	Does not exist

Reusability	All the components are microservices	Only the utilities are reusable

6.3 Evaluating MicroPRA Using Quantitative Methods (MM4S)

Based on the MM4S model elaborated in Section 2.4.4, Table 6.2 shows that there are no direct service-to-service calls in MicroPRA. The metrics in Table 6.2 are based on the dependencies between the services and the number of direct calls between the services. MicroPRA has no inter-service dependencies and no direct service calls because of its event-based nature. MicroPRA satisfies the microservice architecture in other aspects as well, as each business capacity is exposed as a microservice. MicroPRA utilises more fine-grained services, which are aligned with the business capacity. Thus, the MM4S service properties show the achievements and improvements in terms of having loosely coupled services in the MicroPRA.

Table 6.2 Descriptive Comparison of Viscovery and MicroPRA Based on MM4S

MM4S Metrics	**MicroPRA**
Direct service calls (ADS)	More fine-grained services decoupled by the message broker. There are no direct service calls at all.
Dependency for other services (AIS)	Just emits events; no dependency at all.
Number of bidirectional calls (SIY)	No direct calls.

Table 6.3 quantitatively compares MicroPRA and Viscovery in terms of MM4S metrics. However, since information about Viscovery's core components was not available from the literature, this study made some assumptions to perform a comparative analysis. Assumptions made before analysisng the dependencies are the following:

1- Viscovery introduced ingestor, preprocessor, processor and indexer components. This study assumed that each component is a microservice. These micro services are the main building blocks of Viscovery.

2- Novaviz API gateway introduced by Viscovery is an aggregator for utilities such as NLP. Thus, those utilities will be considered as one service in this study.

Table 6.3 Quantitative analysis for MicroPRA and Viscovery based on MM4S

MM4S Metrics	MicroPRA	Viscovery
ADS	0	0 (based on the literature, main components are decoupled)
Dependency for other services (AIS)	0	1 (all components are dependent to Novaviz API gateway to function. We assume that the API gateway is just one compoent)
Number of bidirectional calls (SIY)	0	0

MicroPRA leverages event-based mechanism and there is no direct service call. Therefore, there is no dependency to other microservices. By emitting events, each microservice informs the others about the changes in the system. Table 6.3 shows that MicroPRA exhibits some improvements in achieving better coupling than Viscovery based on the MM4S metrics.

However, in this comparison, only the dependencies between the core services were considered. There are some utility services in MicroPRA and Viscovery such as NLP, POS-Tagger service, were not evaluated in relation to the MM4S metrics.

6.4 Summary

The research target for this work was to study and improve the existing architecture for PRA systems. The prototyped system was implemented based on the proposed event-

based microservice architecture called MicroPRA. Despite some limitations, the qualitative perspectives that were used to evaluate the outcomes, show improvements in terms of different quality attributes. In terms of the descriptive metrics, utilizing of the supplementary components and the event-based approach improved MicroPRA system. Applying some microservice architecture patterns such as event-based architecture, service registry, and API gateway makes the microservices more autonomous and independently deployable.

Even though some prerequisites for doing effective quantitative comparsion between MicroPRA and the existing systems is not fullfiled, using a quantitative quality model which relies on MM4S; and analyses shown that MicroPRA has achieved most improvements, in terms of having weakly-coupled microservices.

CHAPTER 7: CONCLUSION

7.1 Introduction

This chapter is organised into two sections, First, the achievements of the objectives are explained, followed by elaboration on the limitations of the existing work and suggestions and ideas for future works.

7.2 Contributions and Achievement of the Objectives

This research began with studying of the existing PRA systems. Afterwards, the existing literature was reviewed from an architectural perspective. Among the literature, Vsicovery was focused on as it inclines towards the microservice architecture. The purpose was to improve the architecture by applying some design patterns. Evaluating the quality of the MicroPRA architecture relied on some quality models meant for service-oriented architecture, particularly microservice-based architecture.

Based on the research objectives mentioned in Section 1.4, new quality attributes were brought into PRA systems by moving toward microservice architecture. That is why the prototyped system is called MicroPRA. Examining the objectives individually provides a clearer picture of the outcomes.

1- To propose a new architecture, based on the microservice-based architecture for PRA systems that reduces coupling between the components.

   The MicroPRA microservices are loosely coupled compared to the similar system of Viscovery. The facts and figures based on the qualitative criteria show that the coupling is decreased in the MicroPRA due to the event-based system utilised.

2- To implement a prototype PRA system based on the proposed microservice-based architecture as a proof of concept.

The implemented system was used to grab reviews about iPhoneX from Amazon and Lazada. The analyses for this case study shows that MicroPRA works and performs the targeted tasks.

3- To evaluate coupling of the implemented prototype PRA system based on the selected metrics to achieve loosely coupled modules or services as compared to existing systems.

The CoCoME model was used in this study to identify microservices. It was used for the systematic decomposing of the defined sub-domains into microservices. MM4S was used for measuring the achievement of improving maintainability. Table 6.2 shows improvements in the MicroPRA as compared to Viscovery. Another key factor is that the MicroPRA is an event-based system and makes use of a message broker.

The descriptive analysis also involves the use of a Directed Acyclic Graph (DAG) to show the extent to which coupling is weaker than Viscovery. A Microservice registry was also introduced in the MicroPRA. The registry provides much more reliability and scalability for the system as it keeps track of the existing tasks and available microservices that exist. The registry supplies this information to the API gateway, and the API gateway does load balancing based on the availability. Thus, MicroPRA improves the system's availability, reliability, and scalability.

The major contributions of this research are outlined as follows:

1- Some identified PRA architectures were extracted from the existing studies. Evolution of the architectures was analyzed and the research gap in the literature was identified.
2- A new architecture with lower coupling based on microservices utilising an event-based mechanism was proposed.

7.3 Limitations and Future Work

Some of the limitations and future work of this work are as follows:

1- This work does not fully utilise event sourcing such as event repository or event log. Using event sourcing would allow the system to move, follow, and keep track of the histories for each review. By storing events, it would be possible to recreate entities based on the events. For instance, from the time that the review is collected, filtered and processed, all these events would be logged, and the system would rely on the events to generate core data and entities. Adding an event log feature would be beneficial. Keeping an event history would help replay and regenerate data in case of a failure. So, this feature would improve the system with regard to fault tolerance and reliability.

2- Of the service properties proposed by MM4S, only coupling was used in this study. Covering the rest would enable the PRA system to testify against this quality model, and quality assurance would be much more comprehensive.

3- Only the POS tagging feature from the StanfordNLP was used in the MicroPRA. Using more algorithms from this library could bring much more analytical values.

4- Due to a variety of resources for reviews and user content, working on adapter patterns would bring much more flexibility to the system and facilitate its adaptation to different E-WOM sources.

5- Aggregating the data would make the final visualisation and statistical results much faster and resilient. For this purpose, technologies such as GraphQL can be utilised.

6- Bringing in machine learning and big data techniques might require some additional considerations in the proposed architecture.

REFERENCES

Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A., & Lynn, T. (2018). Microservices migration patterns. *Softw Pract Exper*, 1-24.

Bahatia, S., Chaudhary, P., & Dey, N. (2020). Opinion mining in information retrieval. *Springer*.

Bogner, J., Wagner, S., & Zimmermann, A. (2017). Towards a Practical Maintainability Quality Model for Service- and Microservice-based Systems. *Association for Computing Machinery, ACM ISBN 978-1-4503-5217-8/17/09*.

Brown, S. (2014). *Software acrhitecture for developers*: Leanpub.

Bucur, C. (2015). Architecture of a sentiment analysis platform. *Information society and sustainable development*.

Cambria, E., Das, D., Bandyopadhyay, S., & Feraco, A. (2017). *A Practical Guide to Sentiment Analysis*: Springer.

Chen, M., & Sun, Y. (2017). *Sentimental Analysis with Amazon Review Data*. Retrieved from Stanford University: http://cs229.stanford.edu/proj2017/

Chirotti, S. E., Reilly, T., & Rentz, A. (2018). Tracking and controlling microservice dependencies. *Communicattions of the ACM, 61*, 98-104.

Eirinaki, M., Pisal, S., & Singh, J. (2011). Feature-based opinion mining and ranking. *Journal of Comupter and System Siences*,78, 1175–1184.

Espinoza, I., Mendoza, M., & Ortega, P. (2018a). Viscovery: A Platform for Trend Tracking in Opinion Forums. *WISDOM'17, August 2017, Halifax, Nova Scotia, Canada*.

Espinoza, I., Mendoza, M., & Ortega, P. (2018b). Viscovery: A Platform for Trend Tracking in Opinion Forums.

Flory, L., Kweku-Muata, Osei-Bryson, & Thomas, M. (2017). A new web personalization decision-support artifact for utility-sensitive customer review analysis. *Decision Support Systems, 94*, 85.

Francesco, P. D. (2017). Architecting Microservices. *IEEE International Conference on Software Architecture Workshops*, 224-229.

Galin, D. (2018). *Software quality concepts and practice*: IEE Press.

Granchelli, G., Cardarelli, M., Francesco, P. D., Malavolta, I., Iovino, L., & Salle, A. D. (2017). Towards Recovering the Software Architecture of Microservice-Based Systems. *IEEE International Conference on Software Architecture Workshops (ICSAW)*, 46-53.

Gerald, P. (2019). Opinion mining in Web 2.0. Springer Gabler.

Jhipster. (2021). Generate a full stack Java + Angular/React application in a few steps. Retrieved January 21, 2021, from https://www.jhipster.tech/

Hu, M., & Liu, B. (2004). Mining Opinion Features in Customer Reviews. *American Association for Artificial Intelligence*.

Kafka. (2019). Apache Kafka, a distributed streaming platform. Retrieved from https://kafka.apache.org/uses

Pozzi, F. A., Fersini, E., Messina, E. & Li, B. (2016). *Sentiment Analysis in Social Networks*. Morgan Kaufmann.

Killalea, T. (2016). The Hidden Dividends of Microservices. *Communications of the ACM, 19*, 42-45.

Lewis, J., & Fowler, M. (2014). Microservices, a definition of this new architectural term. Retrieved from https://martinfowler.com/articles/microservices.html.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics:System Demonstrations*, 55–60.

Mayer, B., & Weinreich, R. (2018). An Approach to Extract the Architecture of Microservice-Based Software Systems. *2018 IEEE Symposium on Service-Oriented System Engineering*, 21-30.

Mitchell, B. S., & Mancoridis, S. (2006). On the Automatic Modularization of Software Systems Using the Bunch Tool. *IEEE Computer Society*.

Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). Microservice Architecture: Aligning Principles, Practices, and Culture. *O'Reilly*.

Petz, G., Karpowicz, M., Fürschuß, H., Auinger, A., Strítesky, V. c., & Holzinger, A. (2014). Computational approaches for mining user's opinions on the Web 2.0. *Information Processing and Management, 50*, 899-908.

Rihcardson, C. (2019). *Microservice patterns with examples in Java*: Maning.

Robson, K., Farshid, M., Bredican, J., & Humphrey, S. (2013). Making sense of online consumer reviews: a methodology. *International Journal of Market Research, 55*.

Candela, I., Bavota, G., Russo, B., & Oliveto, R. (2016). Using Cohesion and Coupling for Software Remodularization: Is It Enough? ACM Transactions on Software Engineering and Methodology, 25(3), Article 24.

Senivongse, T., & Puapolthep, A. (2015). A Maintainability Assessment Model for Service-Oriented Systems. *Proceedings of the World Congress on Engineering and Computer Science*.

Subramanian, V. (2020). Agile Learner. Retrieved from https://www.agilelearner.com/

Tempero, E., & Ralph, P. (2018). A framework for defining coupling metrics. *Science of Computer Programming*, 214–230.

Tsirakis, N. (2017). Large scale opinion mining for social, news and blog data. *The Journal of Systems and Software, 127*, 237–248.

Tsirakis, N., Poulopoulos, V., Tsantilas, P., LTD, P., & Varlamis, I. (2015). A platform for real-time opinion mining from social media and news streams. *IEE Trustcome*.

Tyszberowicz, S., Heinrich, R., Liu, B., & Liu, Z. (2018). Identifying Microservices Using Functional Decomposition. *Dependable Software Engineering. Theories, Tools, and Applications. SETTA 2018, 10998*.

Zablocki, J. (2015). *Couchbase essentials*. Packt Publishing.