Quality attributes in patterns related to microservice architecture: a Systematic Literature Review

José A. Valdivia*, Xavier Limón†, Karen Cortes-Verdin†
School of Statistics and Informatics, Universidad Veracruzana
Xalapa, Ver; Mexico
*zs15011636@estudiantes.uv.mx, †hlimon, kcortes@uv.mx

***Abstract*—Microservices is an interesting option for those who want to migrate their systems to improve performance, maintainability, scalability, and interoperability. Microservice architecture is a collection of self-sufficient services working together to provide functionalities. Nowadays, there are many options to build microservices, some of them are lead by patterns. However, the mapping between quality attributes and patterns is not clear yet. This systematic literature review presents a microservice pattern collection, it describes their benefits and the association between patterns and quality attributes. Finally, some metrics of quality attributes are identified.**

***Index Terms*—Microservices; Patterns; Quality attributes; Software architecture;**

I. Introduction

Since 2012 large companies such as Amazon, SoundCloud, and Netflix, have started the trend to migrate their systems to a new distributed architecture that is now known as microservices [1], [2], [3]. This architecture still does not have a clear definition in the literature. However, a microservice architecture (from now on MSA) can be understood as a set of small services with precise tasks that through common communication channels interact to achieve business goals. The main microservices benefits are: services with high granularity, easy replacement between services, support for different technologies, decentralized information, and high failure tolerance [4]. Some of the benefits can be traced back to the Service Oriented Architecture (SOA), since microservice architecture can be considered as a specific implementation of it [5].

MSA main structures provide the benefits already mentioned. However, there are structures that can vary depending on the problems faced by the system, and its context. The amount of service instances, the variety of services, and the absence of middleware increase the complexity and introduce the need for service state tracking, services discovery, and distributed events registration [6]. One way to solve these difficulties is introducing patterns in the design. Patterns are a solution of a common problem based on previous experiences to avoid reinventing a solution [7]. Also, patterns expose solutions with reusable structures for similar problems, and provide a vocabulary for the software design community. The pattern structures conform an extension of the architectural elements used for the construction of a system, in the same way enable the construction of more complex systems [8].

Every system pursues the satisfaction of certain quality attributes, which should be a measurable property to assess the stakeholders satisfaction [7]. The inclusion of a pattern in the architecture of a system leads to the consideration of trade-offs for each solution [9]. In other words, decisions on architectural problems should consider the balance between the set of quality attributes desired in the system, and the benefits offered by the proposed solution. For example, *gatekeeper* pattern improves security through the establishment of an entry point, and *competing consumers* enhances performance handling dynamic workloads. Nevertheless, the isolated implementation of *gatekeeper* pattern can negatively affect performance but the combination with *competing consumers* has a positive effect. This is an example of how a pattern decision and combination can imply trade-offs to the quality attributes desired in the system. A metric reduces the difficulty to quantify a property of the system providing a numeric value as a measure of how close an attribute is from the desired satisfaction level [10]. In respect of patterns, the value obtained by a metric enables to compare and select the correct pattern to satisfy the defined goals, and to estimate how impact this decision to other quality attributes.

More precise definitions about architectural patterns, and design patterns can be found in the literature, for example [11] defines architectural patterns as fundamental structures of the system which maintain responsibilities and rules; whereas design patterns are described as subsystems or components added to the system. Also the degree of description is more detailed for design patterns, and its inclusion should not affect the fundamental structures of the architecture. In this work, the term pattern is used interchangeably to refer to design, or architectural patterns. The objective of this work is the compilation of both types of patterns associated to quality attributes in MSA context, also the identification of metrics used to measure the fulfillment of one quality attribute related to a specific pattern. This work does not intend to present a classification between the patterns found in the literature.

In regard to microservice architecture in the literature, as far as we know, the pattern collections in microservices area is limited. However, we can find studies like [12], and [13] with numerous architectural patterns providing a base of alternatives. However, such studies lack of mapping between quality attributes, and patterns. Therefore, it is difficult to identify which pattern is the appropriate to satisfy a quality

978-1-7281-2524-4/19/$31.00 ©2019 IEEE
DOI 10.1109/CONISOFT.2019.00034

attribute. Also it is complex to identify which quality attributes could be affected by the selected pattern. Even to this day, no study has been found exposing metrics for the measurement of a quality attribute related to a pattern in the context of MSA.

The architectural, and design patterns in microservices are important elements for the construction of software, since the inclusion of a pattern in the architecture provides a reliable solution to a problem, improves communication across the development team, and even helps to satisfy one or more quality attributes. However, identifying available patterns for microservices is a task that requires time, and effort; even more if the benefits of the patterns are too similar and it is necessary to carefully select among others patterns that give solutions to the same problem.

Therefore, the aims of this study are to describe through a systematic review of the literature the patterns used in microservices, identify the trade-offs in selecting a pattern, either design or architectural, and associated the quality attributes implicitly related. This paper also aims to identify metrics that allow to quantify the degree of satisfaction of a quality attribute related to a pattern. In this way, it is expected to reduce the time for software architects to identify and select the microservice patterns that best fit their needs, also to provide a pattern collection improving the literature in microservices area.

This document is divided into the following sections: Section II reviews the background on systematic literature reviews in microservice area, and exposes the difference between the studies so far and the present study. Section III explains the systematic method for collecting information, its considerations, limitations, and inclusion and execution criteria. Section IV presents the results of the information extraction on the identified patterns and the data interpretation. Section V concludes the document with the most relevant findings, and proposes opportunity areas to investigate.

II. Related work

Despite the microservices trend has emerged in the industry since 2014, few academic literature has been published compared to the informal literature from the industry. As far as we know, between 2014 and 2018 only four secondary studies have been published. A secondary study is guided for at least one research question and a review protocol to search and select the information; the result is a collection of the most relevant primary studies regarding the research questions, finally the analysis of the collection of the primary studies provides the answer to the research questions; meanwhile a primary study does not gather any other study and contributes directly about one topic. The first secondary study about microservice architecture was published in 2016 [14]. The aim of this study was to identify trends and research maturity on microservices. The contributions of this study benefited to researchers detecting the areas of interest and how the information was published. This study was merely an exploration of microservices literature. They found even useful to include blogs to extract the industry perspective of microservices principles. The overview of 21 selected studies describes software engineering as the main computer field that contributed. Other highlights from the overview were: most of the publications provide a solution or validation to the architecture or method, six design patterns were identified, and the most frequent publication format were magazine article and thesis: this leads to the immaturity of the microservices literature.This was a good first secondary study because it is not specific to one topic, it introduces a microservices summary, analyzes the beginnings of the MSA literature and identifies posible future research; however, the identified design patterns are only mentioned and it did not provide any relation with quality attributes or metrics. In 2016, [15] introduces more specific perspective in microservices, they examine the architectural area, which was not discussed in the first study. They analyzed the literature to identify the main challenges in microservices, the architectural diagrams or views used, and the quality attributes related to MSA. From the 33 primary studies they identify that communication and integration, and deployment operations were the main microservices challenges. The solution proposal and the validation research documents provides a selection of diagrams used to explain the proposal design, the dominant types were component diagram, and context diagram; they also found a justification absence to use other diagrams, this could indicate a need for microservices modeling language. Finally, the main quality attributes related to the architecture are scalability, independence, and maintainability, however this contributes only to the general architecture and the patterns are not a subjects of study. In 2017, [16] supplements the last two studies regarding the type of research conducted, and examines practical motivations to carry out a MSA research. They identify from 37 studies emerging standards, and trend tools (technologies) for microservices. The most frequent type of research was the same reported for [15], but this study contributes classifying it by service type, the two most common were Software as a Service(SaaS), and Platform as a Service(PaaS), unfortunately almost half of the documents does not mention the service type. The main practical motivations to perform primary studies were the functionality related to SaaS and PaaS, and performance related with less frequency to all the service types. The most frequent standard by far was REST, on the other hand the tools was a broadly reported area, this encompasses 22 tools or technologies where the most frequent were Docker, Node.js, and MySQL. This study and [15] contribute to the microservice overall, as well as, this study filled a specific lack about standards and technologies also this study has been the only one with a systematic literature review approach; however the patterns topic is not covered. Finally, in 2018 [12] integrates more information about the principles of the microservices style from 42 primary studies, this study contributes to the principles identified in [14] detailing the advantages and disadvantages in microservices. They explore also the area of architectural patterns; the result was a nine patterns catalogue, each pattern was described by a template, which included: Concept,

origin, properties, evolution, reported usage, advantages, and disadvantages. The catalogue is divided in three categories, and the patterns included were:

- Orchestration and coordination: Patterns for communication and coordination from a logical perspective.
  - API-gateway
  - Client-side discovery
  - Server-side discovery
  - Hybrid
- Deployment strategies: Physical strategies and virtual machines
  - Multiple service per host
  - Single service per host
- Data storage: Data management in addition to inter-component communication.
  - Database-per-service
  - Database cluster
  - Shared database server

Our study differs of [12]: conducting a systematic literature review and not a mapping study, in this way we analyzed the patterns topic in MSA context, and improved the protocol using five criteria stages, and the reciprocal translation method enhancing the synthesis process; to reduce the risk to include not validated information we did not include grey literature; design patterns were included to the collection; a mapping between patterns and quality attributes was made. Nevertheless, our study has similarities, we include a snowballing method in the review process, also five of the eight bibliographic sources were included and both studies enrich the temporal results on patterns collection.

III. Review process definition

Few reviews on microservices have been identified in the academic literature, and only one of them has been published under the modality of systematic review of the literature. Since the microservices trend is relatively new is not common to find secondary studies. Although, these studies have improved the area of microservices, also have provided a selection of gaps to be covered by new primary studies.

The systematic reviews are described by [17] as studies where relevant information is identified, and evaluated with respect to a question or topic. It summarizes the evidence related to a topic, helps to identify areas of research, and formalizes a collection of studies as background on an emerging topic.

Therefore, the present study conducted a systematic review with the guidelines for performing systematic literature reviews in software engineering [17]. As this guideline suggests, a systematic search strategy with inclusion, and exclusion elements was established to identify the most relevant studies in the literature. Also the process is guided by three research questions about design, and architectural patterns in microservices area.

RQ.1 What patterns are identified in microservice systems?

TABLE I
MICROSERVICES KEYWORDS AND RELATED CONCEPTS USED IN THE SEARCH QUERY

Keyword	Related concepts
Microservices	Service-oriented architecture, micro-service*, micro service*, microservices architecture
Architectural pattern*	Pattern*, architectural style, architectural tactic
Quality attribute*	Quality, architectural driver*, quality, requirement*, QA, quality attribute requirement*
Metric*	Response measure, measure*, checklist, analytic model, instrumentation, evaluation

RQ.2 Which quality attributes are related to the patterns used in microservices?

RQ.3 What metrics are used to quantify quality attributes related to a microservices pattern?

The **RQ.1** aims to identify the microservices patterns in the literature or patterns used in related architecture styles that satisfy a MSA need, for example, through SOA. The **RQ.2** represents the predominant interest in this study because it will help to understand why these patterns are used in certain problems, likewise, a comparison between the patterns that are used to achieve the same quality attribute or to resolve the same problem. Finally, in regard to quality attributes, it will be possible to define which are the most desired ones for microservices systems. The **RQ.3** reflects the interest of exploring the metrics to support the selection of patterns. The purpose is to identify metrics to evaluate each quality attribute related to a microservices pattern in order to facilitate the measurement of quality attribute satisfaction.

A. Protocol

To gather the information, automated searches were used in Scopus indexing services, and in the following repositories: Springer Link, Science direct, Association for Computing Machinery (ACM), and IEEE Xplore. These sources were chosen due to their relevance in software engineering and technology areas, and the access provided by CONRICyT. To delimit the results, the following criteria were applied for the automated search and for the four selection stages for the review.

Inclusion criteria:

- Publication between January 2014 to September 2018. The start date was determined by the year of the first microservices study in the literature, and the end date was defined according to the limited time of the search process.
- Studies written in English. This language is the most common in the technology literature, besides is the most fluent language by the researcher team excluding native language.
- Arbitrated studies. To enhance the quality of the study, the primary studies must have been accepted to be published by third parties

Exclusion criteria:

- Opinion, anecdotal or personal experience document type. The materials that could influence the merit of the information were not considered to improve the impartiality of the results.
- Book chapter or book. Due to the limitations of resources, and time, material with this extension was not considered.

For this automated search we identified the keywords from the last microservices reviews and we added related concepts for each main keyword as is shown in Table I. To cover the three research questions we combine three special queries, each one was associated with a research question. The following string is associated to **RQ.1**:

("Microservices" OR "Service-oriented Architecture" OR "Micro-service*" OR "Micro Service*" OR "Microservices Architecture") AND ("Architectural Pattern*" OR "Pattern*" OR "Architectural Style" OR "Architectural Tactic")

RQ.2:

("Quality Attribute*" OR "Quality" OR "Architectural Driver*" OR "Quality Requirement*" OR "QA" OR "Quality Attribute Requirement*")

RQ.3:

("Metric*" OR "Response Measure" OR "Measure*" OR "Checklist" OR "Analytic Model" OR "Instrumentation" OR "Evaluation")

The result was a single search query with the last three search strings, this final search query included all the keywords related to the aims of this study and it was combined to logically include any possible study related to any research question. In addition to the automated search with the previous criteria and search query, a manual research was incorporated using the backward snowballing method [18]. In this case, the inclusion criteria for snowballing was to be referenced in at least two studies, and fulfill all the previous requirements. Finally, to synthesize the gathered information we used a meta-ethnographic method: reciprocal translation [19].

B. Review process

The selection of studies was made in four stages:

1) Exclusion by title
2) Inclusion by snowballing
3) Exclusion by checklist
4) Exclusion by answer to research questions

First, an automated search was executed to the five information sources, recovering 366 studies matching the search strategy filters as is show in Fig. 1. After that, the exclusion for stage 1 discarded any document title not related to a research questions, this reduced the number of studies to 24. Then, for the previous studies, in the stage 2 we found four studies from the 18 studies retrieved from Scopus, yielding a total of 28 studies. Although, due to the possibility of answering a research question, an exception to the inclusion criteria was made for three of the four studies found, three studies were cited by only one document. The checklist for the stage 3 helped to identify the studies related to a research question and to evaluate the paper quality, the elements considered were:

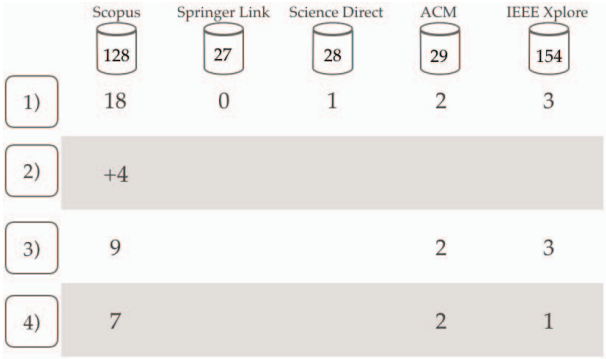

Fig. 1. Repository frequency agains review process stage

- Does the text potentially answer a research question?
- Does the text demonstrate clarity in the purpose of the investigation?
- Was the investigation validated?
- Does the study focus specifically about patterns in microservices?

The abstract of each paper was read to verify that at least three of previous four points of the checklist were met. After the exclusion from the checklist the total number of studies was reduced to 14. Finally, in the stage 4 a complete reading of the studies was carried out, and the answer of at least one research question was verified by another checklist. The final result was 10 studies for information extraction. It is important to highlight that information extraction consisted of filling a table about the general paper information, and other special table for each answer to a research question, giving a total of 32 tables with information. With regard to the synthesis process, we identified sixteen concepts from the last 32 tables, then the information was grouped by concept and ordered chronologically ascended fashion.

IV. Results

Table II shows the ten final documents analyzed, these are sorted alphabetically by title. The analysis of these ten documents give a perspective on the interest of the community for the design, and architectural patterns area as a way to solve problems in microservices. The frequency of the selected studies of related work is point out in Fig. 2. We compare the four previous secondary studies and our study regarding the MSA trend in the literature. It is interesting that one of the studies found one document related to MSA in 2010, nevertheless the authors confirm that the document does not have a direct relationship with what is currently known as MSA because the characteristics described are not the same identified in MSA. Also, the interest in MSA exists since 2014 and it has a high frequency in 2015, even though

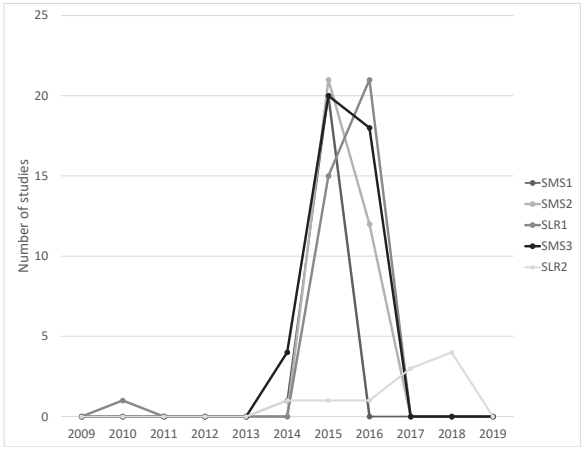

Fig. 2. Microservice architecture, and microservice patterns trend reported by secondary studies. The secondary studies are presented in a chronologically ascending order and secondary study type. SMS1 = [14], SMS2 = [15], SLR1 = [16], SMS3 = [12], RSL2 = the present study.

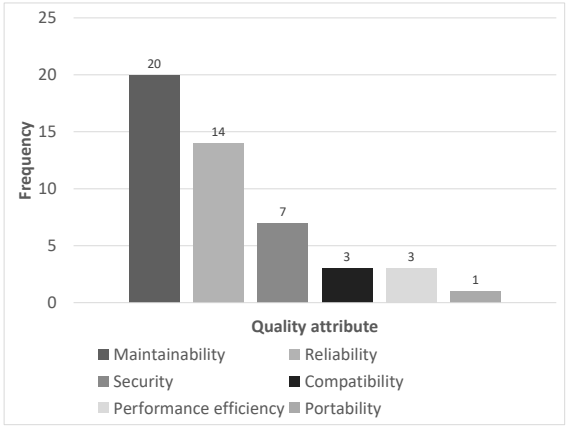

Fig. 3. Quality attributes frequency related to the identified microservice patterns

the microservice patterns topic is almost infrequent in the literature but slightly is increasing since 2017. Analyzing the extracted information from our study, the origin of the documents in this area are of most interest to researchers in Germany, followed by Canada, and USA.

Table III presents the 34 patterns identified and the architecture where each pattern was proposed or implemented. To identify the architecture we mapped the study title and the context. For all the cases it was clear to identify the architecture related to the pattern, even though this not confirm the pattern origin or its exclusivity. Despite the number it should be considered that some of the patterns are specialized implementations of other patterns, for example *service registry client* is a specialized implementation with tactics and a dedicated unit of *service registry*. Some patterns have a simple implementation, and therefore their abstraction degree, and scope could differ from what would be expected from a pattern, for example, *correlation ID* helps to refine the request identification, so the complexity is low compared other patterns. Also, we identified that two patterns are under different terms but they have the same principles: *Change code dependency to service call* with *adapter microservice*, and *load balancer* with *load-balancing*.

With regard to quality attributes we used as a reference the product quality model defined in ISO/IEC 25010 [29] . This model comprises eight quality characteristics, however we found only six characteristics associated to microservice patterns: Reliability, performance efficiency, security, compatibility, maintainability, and portability; we did not find any association with functional suitability, and usability. Fig. 3 shows the frequency of the attributes, some of the patterns are related to more than one attribute. The three most frequency attributes are maintainability, reliability, and security, but the frequency between second and third place differs for the double. Finally, the identified metrics associated with quality attributes in patterns are:

- Time
- Percentage of requests accepted
- Number of files impacted for modification (maintainability)
- Energy consumption

The following subsections present the detailed results of each research question. The first subsection describes the collection of the identified microservice patterns, the second subsection indicates the relationship between each pattern and quality attributes, and the last subsection analyzes the metrics for quality attributes related to patterns.

A. ***RQ.1*** *Analysis*

In relation to **RQ.1**, Table III presents the patterns identified in the literature in the area of microservices. It is worth mentioning that some of the patterns are related to SOA, which could be considered as a more general implementation of a distributed service architecture [5], for example, *secure channel, asynchronous query, service locator, asynchronous completion token* and *event notification*. The association can be identified by the characteristics present in microservices from SOA. This is a non-centralized architectural style, and its principle is communication between services to offer functionality including consuming third-party services [30]. However, SOA has challenges in communication, middleware, and granularity; although these challenges have been tried to mitigate with the emergence of microservices [5].

The most frequently identified patterns are related to solve communication problems, and coordination between services, for example, *competing consumers, priority queue, API gateway,* and *backend for frontends*. Besides, the term design or architectural pattern still remains not clear among authors, which causes some patterns to be classified in both categories or even in some cases under the general pattern term. For

TABLE II
DOCUMENTS SELECTED

Number	Study title	Format
1	A case-based reasoning approach to reuse quality-driven designs in service-oriented architectures [20]	Article
2	An Empirical Study of the Impact of Cloud Patterns on Quality of Service (QoS) [21]	Conference proceedings
3	Poster: Exploration of Academic and Industrial Evidence about Architectural Tactics and Patterns in Microservices [13]	Conference proceedings
4	Guidelines for adopting frontend architectures and patterns in microservices-based systems [22]	Conference proceedings
5	Highly Scalable Microservice-based Enterprise Architecture for Smart Ecosystems in Hybrid Cloud Environments [23]	Journal article
6	Implementation Patterns for Microservices Architectures [24]	Conference proceedings
7	Incorporating Security Features in Service-Oriented Architecture using Security Patterns [25]	Journal article
8	Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications [26]	Conference proceedings
9	Microservices migration patterns [27]	Journal article
10	Understanding the impact of cloud patterns on performance and energy consumption [28]	Journal article

TABLE III
MICROSERVICE PATTERNS IDENTIFIED IN THE LITERATURE AS MICROSERVICES SOLUTIONS OR INFERRED FROM OTHER APPROACH. MICROSERVICE ARCHITECTURE (MSA) AND SERVICE ORIENTED ARCHITECTURE (SOA) ARE ARCHITECTURES WITH SIMILAR CHARACTERISTICS; HOWEVER, IS INTERESTING THAT CLOUD ARCHITECTURE (CA) IS ALSO RELATED WITH MSA PATTERNS.

Pattern name	Architecture
API gateway	MSA
Asynchronous completion token	SOA
Asynchronous query	SOA
Auth-service	MSA
Backend for frontend	MSA
Change code dependency to service call/adapter microservice	MSA
Circuit breaker	MSA
Competing consumers	CA
Container	MSA
Correlation ID	MSA
Deploy cluster and orchestrate containers	MSA
Edge server	MSA
Event notification	SOA
External load balancer	MSA
Externalized configuration	CA
Gatekeeper	CA
Internal load balancer	MSA
Key-Value store	MSA
Load balancer/load-balancing	MSA and CA
Local database proxy	CA
Local sharding-based router	CA
Log aggregator	MSA
Microservices DevOps	MSA
Page cache	MSA
Pipes and filters	CA
Priority queue	CA
REST integration	MSA
Results cache	MSA
Secure channel	SOA
Self-containment of services	MSA
Service discovery	CA
Service locator	SOA
Service registry	MSA
Service registry client	MSA

example, the following terms were found under the pattern category: *microservice architecture, pipes and filter* [13], *and container* [28]. Microservices architecture should not be considered as a pattern because this represent an architecture style which can encompasses several design patterns; *pipes and filters* is a complex term in the literature, for [31] is considered as architecture style meanwhile for [7] is an architectural pattern, therefore is not clear the abstraction degree; *container*, is a specialized implementation from a virtualization architecture style [30], consequently the abstraction degree is limited by the technology implementation.

The following subsections analyze the benefits, and disadvantages of each pattern. For ease of interpretation the patterns have been grouped, therefore the following groups do not intend to be a classification proposal. The description of each grouping of patterns begins with two elements. First the benefits shared among the grouped patterns are described, then the disadvantages are described by comparing each pattern with the other. Each pattern will be presented in a chronologically ascending order.

1) Data persistence: Patterns oriented to improve the management of the system database allowing information independence, and scalability. The shared benefits are: non-blocking producers in case of failure, and dynamically change of instances. However, these patterns can increase the complexity of the system or have limitations in complex queries [21] [28].

In 2014 *local database proxy*, and *local sharding-based router* appeared in [21], the first pattern is useful for applications that have heavy readings loads, but with limitations for a large number of write operations, whereas the second pattern endures a large number of writing, and reading operations. In 2018 more information is included in [28] about both patterns. The *proxy* pattern provides elasticity at the execution time since its master and slave principle allows to add or remove instances. On the other hand, in *router* the sharing logic can be achieve by multiple strategies, the work can be balanced through the shards even.

2) Communication: The patterns in this group improve communication, and coordination between services or provide a communication channel. This improve services identification, and reduce complexity in large systems. nonetheless, these patterns may cause unnecessary complexity for small

systems, single point of failure, or these cannot be a final implementation.

Patterns start from 2015, and during the three following years maintain a constant documentation in the academic literature. *Secure channel* was the first pattern found in [25], which generally allows a secure communication channel for services. The next year, in [24] the patterns described were *adapter microservice*, and *service registry*. The first one creates a service as an intermediary to incorporate segments that are not services yet, it can be a simple solution when a system is being migrated. The second pattern enables decoupling a physical location to a service creating a service instance identifier. In 2017 is described *service discovery* in [26], which allows dynamic identification of services through meta-information. Also, tactics can be incorporated to verify the status of registered services. In 2018 are described the *service locator*, and *event notification* benefits in [20]. The first pattern is useful to reinforce security, and the second one for reliability. In the same year, *service registry*, and *change code dependency to service call* are found again in the literature in [13]. The last one can be identified as an adapter counterpart, since it does the same but with a different objective. Also, this pattern is described as a solution to avoid dependencies, and isolate the code in a service. Besides *REST integration* is mentioned as a solution to include communication points between services. The same year in [27] *service registry* is again mentioned, and *service registry client* is included as a variant, this one could implement tactics to improve service control. One tactic example is heartbeat which confirm the status of the service, and support to decide if is necessary to keep it in the registry. Finally in 2018 in [28], *competing consumers* enables the management of flexible workloads through the deployment and coordination of consumers. In addition, the same study recommend the use of *pipes and filters* for the decomposition of complex processes, and the improvement of reliability because if a part fails it can be reprogrammed to complete the task.

3) Entry point: The following patterns are oriented to offer a control, and entry point to a service or a type of backend. However, all patterns have the disadvantage of being a single point of failure, and be inconvenient for high workloads.

The first pattern was identified in 2016 in [24]. *Backend for frontend* creates a special backend for each client type. In this way the clients can be grouped, and redirected to a specific channel. Also this pattern is useful as a verification point since all the services must be related to at least one consumer type. In 2017 three studies the [26], [22], and [23] describe the *API gateway* advantages: a solution to authentication, to hide the real instance from the service client, to monitor the workload, and load balancing. Actually it is the most frequent pattern in the analyzed literature. Other patterns were described in the first two of the three previous studies. The first study confirms the importance of *backend for frontend*. Other benefits about this pattern are reducing the load of the *gateway*, and to facilitate the connection point for the frontend team at the development stage. The second study mentions *auth-service* to separate information from users with reduced functionality, and to improve the response times of the authorization process. In 2018 *backend for frontend* is identified again in [13]. Finally in another study [28], *gatekeeper* is identified to support applications that handle sensitive information, since it is responsible for verifying identities, requests, and redirecting requests to reliable instances.

4) Distribution: In this subsection patterns help to the distribution of services in a logical way. With exception, in some cases, it may be complicated to fully implement the pattern or it only partially address the problem.

Two studies were identified in the literature. [26] in 2017 presents the *externalized configuration* pattern to maintain main configurations outside the services, and allow only read requests. In [13] four patterns were identified. *Self-containment of services* enhances the autonomy by decoupling services, and gathers the required libraries. Then *container* is mentioned again as a technological option for the compliance of the first pattern, also in this way the resources can be restricted, and likewise the deployment. Following this trend, *deploy cluster and orchestrate containers* continues latest pattern principles, and constitutes the option for managing multiple instances. Finally, in the same study, *microservices DevOps* is described as a pattern for the identification, and development of services. However, most of the latest patterns maintain a granularity degree that could call into question the identification as a pattern.

5) Supplementals: According to the literature the following patterns work best accompanied by another pattern since its implementation is very specific, and solve a single problem. The patterns that work best under some specific implementation fall into the two following cases. The first case, when implementing *backends for frontend* or *change code dependency to service call or adapter microservice* can be implemented:

- *Correlation ID*
- *Log aggregator*
- *Key-value store*
- *Page cache*
- *Results cache*

The second case, if *local database proxy* or *local sharding-based router* is implemented, *priority queue* can be useful. The rest of the patterns that conform the group represent specific solutions to a concrete problem, and given the complexity of the problem it is possible that the inclusion of one these patterns does not solve the problem completely.

The first supplemental pattern was identified in 2014 in [21]. The *priority queue* benefits communication between the components. Two years later in [24] four simple patterns improve the storage, and retrieval of information. The first two patterns, *results cache*, and *page cache* help to improve performance in retrieval of information. Although both patterns have weaknesses, *results cache* may get not up to date information while *page cache* is not appropriate for mobile environments. On the other hand, it is possible to refine the service or request identification by implementing *key-value store* or *correlation ID* with the last two patterns. The pattern *log aggregator*

collects the log files on a single element allowing to keep track of the events in one place, this is helpful to check microservices health. During 2017 the studies [26], and [23] mentioned *load balancing* pattern. This pattern handles several requests with different time frames, also it can be implemented with several algorithms but round robin is the most beneficial. In 2018 *asynchronous query*, and *asynchronous completion token* were identified in [20] these patterns seek to improve performance. The same year *circuit breaker*, and *edge server* were described in [27]. The first pattern is a known pattern in the literature to avoid faults when a service is not available, and it helps the system to contain failures until the service is restored. The second pattern obscures the server structure details to reduce complexity; in this way, dynamic routes can be established, and the information be tracked. Also, two specialties of the *load-balancer / load balancing* pattern previously mentioned in 2017 are described. The *internal load balancer* controls the load of multiple service instances, and holds an entry of them. In addition, local metrics, and algorithms can be implemented to define the best instance. On the other hand, the *external load balancer* usually uses the *services registry* list because its scope is not specific to one kind of instances. Therefore it can not consumes local metrics, but algorithms can be implemented to improve its work. Finally in [28], *priority queue* was identified again. The importance to improve the messages communication with different degrees of priority is reaffirmed.

B. *RQ.2* Analysis

Regarding the **RQ.2** the quality attributes identified in the literature correspond to those related to MSA.Table IV presents the quality attributes associated to each identified pattern. To realize the mapping process we identified directly in the paper the attribute related to the pattern, however not all the patterns reported have a documented association with a quality attribute, for those cases we analyzed the characteristics and consulted a secondary researcher with microservices experience. Furthermore it is interesting that the third quality attribute with more frequency is security. This attribute is not directly related to MSA, however, in the literature the attention it can be noticed by the frequency of patterns to benefit this attribute.

C. *RQ.3* Analysis

Finally, about **RQ.3** the metrics identified in the literature associated with the evaluation of a quality attribute in a pattern are: Time, percentage of requests accepted (interoperability), number of files to be modified (maintainability), and energy consumption.

Three special aspects were identified related to time: the response time to fulfill a number of requests per second, the interaction time (time before the user can interact with the GUI), and the response time. Related to time in 2014 in [21] were analyzed the patterns *local database proxy, local sharding-based router*, and *priority queue*. The three patterns have a positive impact with respect to reading, and writing requests. The first pattern is more suitable for reading request. The second one for writing, and the last one have a moderate positive effect in both. It is also appropriate to combine the last pattern with one of the first two patterns. Besides, the algorithms choice in these three patterns does not represent a big difference since the key point is to make the right pattern decision.

In 2017 *Self-containment*, and *API gateway* patterns were evaluated for modifiability in [22]. In regard to the number of files to modify, the first pattern is the most suitable. Likewise, about percentage of acceptances, the first pattern is better than the second one. The *API gateway* weakness is facing high workloads. Finally both obtained a similar number of milliseconds in low request but in cases with high demand affirming the weakness the second pattern takes a little more time.

The next year *local proxy database, local sharding-based router, priority queue, competing consumers, gatekeeper*, and *pipes and filters* were evaluated in energy consumption in [28].

TABLE IV
QUALITY ATTRIBUTES RELATED TO MICROSERVICES PATTERNS

Pattern name	**Quality attribute**
API gateway	Reliability and Security
Asynchronous completion token	Reliability
Asynchronous query	Reliability
Auth-service	Reliability
Backend for frontend	Compatibility
Change code dependency to service call/adapter microservice	Compatibility
Circuit breaker	Maintainability, and Reliability
Competing consumers	Maintainability
Container	Maintainability
Correlation ID	Maintainability
Deploy cluster and orchestrate containers	Reliability, and Maintainability
Edge server	Maintainability, and Reliability
Event notification	Reliability
External load balancer	Maintainability, and Reliability
Externalized configuration	Security
Gatekeeper	Security
Internal load balancer	Maintainability, and Reliability
Key-Value store	Security, and Reliability
Load balancer/load-balancing	Performance efficiency*
Local database proxy	Maintainability, and Reliability
Local sharding-based router	Maintainability, and Performance efficiency
Log aggregator	Maintainability*
Microservices DevOps	Maintainability*
Page cache	Performance efficiency
Pipes and filters	Reliability, and Maintainability
Priority queue	Maintainability
REST integration	Maintainability, and Compatibility*
Results cache	Performance efficiency
Secure channel	Security
Self-containment of services	Maintainability*
Service discovery	Security and Maintainability*
Service locator	Security, and Maintainability*
Service registry	Portability, Maintainability, and Reliability
Service registry client	Maintainability, and Reliability

* = Not reported but inferred by the literature

In general, the patterns improve reduction of energy consumption, and time in performance efficiency in microservices compared to monolithic systems. *Pipes and filters* improves the performance time, and the reduction of energy consumption. Surprisingly, this pattern does not achieve the same benefits in monolithic systems. Likewise, these benefits can be achieved with *gatekeeper*, and *competing consumers*, but *gatekeeper* isolated implementation can negatively affect response time, and energy consumption. The combination of *local database proxy*, and *competing consumers* results in a negative impact on response time. In the same way does the combination of *local sharding-based router, competing consumers*, and *pipes and filters*.

V. Conclusions

This work contributes to the identification of microservices patterns in the academic literature. Some of these patterns can be traced from SOA, and have been slightly modified to comply with microservices. Furthermore, as a concept, the term pattern differs in the literature, and sometimes requires an analysis to identify whether is a design or architecture pattern. The decision to use a pattern or the combination of them carry out different results in a system. Therefore it is recommended to analyze all the trade-offs for the quality attributes pursued by the system.

It has been identified that several patterns are oriented to support the migration transition; even patterns that are only useful in the migration process and do not represent final solutions are identified. The number of metrics identified in the patterns is low, the information in the academic literature is rare. The metrics found are related to performance efficiency, however, the attention for energy consumption as a metric for the decision of patterns choice is an interesting observation about the new elements to be considered as needs.

Finally, no detailed information about some of the mentioned patterns was found. There are patterns that represent gaps to be covered. The literature on patterns in microservices is an increasing trend. The inclusion of more patterns to the present list is expected, therefor the identified patterns list is transient. It should not be considered as a final list. In regard to the metrics identified, these only cover a small part of the identified patterns. This is a potential area that could improve the pattern selection criteria through a more detailed, and quantitative comparison study.

References

[1] T. Hoff, "Amazon Architecture," 2007. [Online]. Available: http://highscalability.com/

[2] Netflix Technology Blog, "Netflix Conductor: A microservices orchestrator," 2016.

[3] P. Calçado, "Building Products at SoundCloud — Part I: Dealing with the Monolith," 2014. [Online]. Available: https://developers.soundcloud.com/blog/building-products-at-soundcloud-part-1-dealing-with-the-monolith

[4] J. Lewis and M. Fowler, "Microservices - A definition of this new architectural term," pp. 1–16, 2014. [Online]. Available: http://martinfowler.com/articles/microservices.html

[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 1st ed. O'Reilly Media, 2015.

[6] S. Mak, "Modules vs microservices," 2017. [Online]. Available: https://www.oreilly.com/ideas/modules-vs-microservices

[7] L. Bass, P. Clements, and R. Kazman, *Software Architecture In Practice*, 3rd ed. Addison Wesley, 2012.

[8] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects*. Chichester, UK: Wiley, 2000. [Online]. Available: https://www.safaribooksonline.com/library/view/pattern-oriented-software-architecture/9781118725177/

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[10] K. H. Möller and D. J. Paulish, *Software metrics: a practitioner's guide to improved product development*, ser. Chapman and Hall computing series. IEEE Computer Society Press, 1993. [Online]. Available: https://books.google.com.mx/books?id=FvUpAQAAMAAJ

[11] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing, 1996.

[12] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural Patterns for Microservices: A Systematic Mapping Study," *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, no. Closer, pp. 221–232, 2018. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006798302210232

[13] F. Osses, G. Márquez, and H. Astudillo, "Poster: Exploration of academic and industrial evidence about architectural tactics and patterns in microservices," in *Proceedings - International Conference on Software Engineering*. IEEE Computer Society, 2018, pp. 256–257. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85049679642{\&}doi=10.1145{\%}2F3183440.3194958{\&}partnerID=40{\&}md5=5b348ca431f2d7eb64276440699c13ed

[14] C. Pahl and P. Jamshidi, "Microservices: A Systematic Mapping Study," *Proceedings of the 6th International Conference on Cloud Computing and Services Science*, vol. 1, no. Closer, pp. 137–146, 2016. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0005785501370146

[15] N. Alshuqayran, N. Ali, and R. Evans, "A systematic mapping study in microservice architecture," in *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016*. IEEE, nov 2016, pp. 44–51. [Online]. Available: http://ieeexplore.ieee.org/document/7796008/

[16] H. Vural, M. Koyuncu, and S. Guney, "A systematic literature review on microservices," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10409 LNCS. Springer, Cham, 2017, pp. 203–217. [Online]. Available: http://link.springer.com/10.1007/978-3-319-62407-5{_}14

[17] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

[18] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. Citeseer, 2014, p. 38.

[19] G. W. Noblit and R. D. Hare, *Meta-ethnography: Synthesizing qualitative studies*. sage, 1988, vol. 11.

[20] G. Rodríguez, J. A. Díaz-Pace, and Á. Soria, "A case-based reasoning approach to reuse quality-driven designs in service-oriented architectures," *Information Systems*, vol. 77, pp. 167–189, 2018. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85049518388{\&}doi=10.1016{\%}2Fj.is.2018.06.003{\&}partnerID=40{\&}md5=2a263db3db321226f573632f1966b487

[21] G. Hecht, B. Jose-Scheidt, C. D. Figueiredo, N. Moha, and F. Khomh, "An Empirical Study of the Impact of Cloud Patterns on Quality of Service (QoS)," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. IEEE, dec 2014, pp. 278–283. [Online]. Available: http://ieeexplore.ieee.org/document/7037678/

[22] H. Harms, C. Rogowski, and L. Lo Iacono, "Guidelines for adopting frontend architectures and patterns in microservices-based systems," in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, S. W. v. D. A. Zisman A. Bodden E., Ed., vol. Part F1301. Association for Computing Machinery, 2017, pp. 902–907. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85030759214{\&}doi=10.1145{\%}2F3106237.3117775{\&}partnerID=40{\&}md5=1681968c0165204c8c29dd4f2c486ffa

[23] D. Müssig, R. Stricker, J. Lässig, and J. Heider, "Highly Scalable Microservice-based Enterprise Architecture for Smart Ecosystems in Hybrid Cloud Environments," *Proceedings of the 19th International Conference on Enterprise Information Systems*, vol. 3, no. Iceis, pp. 454–459, 2017. [Online]. Available: http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006373304540459

[24] K. Brown and B. Woolf, "Implementation patterns for microservices architectures," in *Pattern Language of Patterns*, 2016, p. 7. [Online]. Available: https://dl.acm.org/citation.cfm?id=3158170

[25] A. K. Dwivedi and S. K. Rath, "Incorporating Security Features in Service-Oriented Architecture using Security Patterns," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 1, pp. 1–6, 2015. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2693208.2693229

[26] K. A. Torkura, M. I. Sukmana, F. Cheng, and C. Meinel, "Leveraging Cloud Native Design Patterns for Security-as-a-Service Applications," in *Proceedings - 2nd IEEE International Conference on Smart Cloud, SmartCloud 2017*. IEEE, nov 2017, pp. 90–97. [Online]. Available: http://ieeexplore.ieee.org/document/8118424/

[27] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, and T. Lynn, "Microservices migration patterns," *Software: Practice and Experience*, no. June 2018, pp. 2019–2042, 2018. [Online]. Available: http://doi.wiley.com/10.1002/spe.2608

[28] F. Khomh and S. A. Abtahizadeh, "Understanding the impact of cloud patterns on performance and energy consumption," *Journal of Systems and Software*, vol. 141, pp. 151–170, 2018. [Online]. Available: https://doi.org/10.1016/j.jss.2018.03.063

[29] ISO, *ISO/IEC 25010: 2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation (SQuaRE)-System and Software Quality Models*. ISO, 2011.

[30] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Wiley Publishing, 2009.

[31] H. Cervantes and R. Kazman, *Designing Software Architectures: A Practical Approach*, ser. SEI Series in Software Engineering. Boston: Addison-Wesley, 2016. [Online]. Available: http://my.safaribooksonline.com/9780134390789