Architectural Languages in the Microservice Era: A Systematic Mapping Study

Anonymous Author(s)

ABSTRACT

In modern software systems, Microservice Architecture (MSA) has gained popularity over monolithic design by providing the ability for flexible and independently upgradable services. Although there are considerable benefits that MSA provides, as new microservices are introduced into these MSA-based systems, they can become increasingly complex and hard to understand. Architectural languages are a potential solution to this problem because they can provide a comprehensive overview of system's architecture as it changes. In this paper, the authors conduct a systematic mapping study to identify the architectural languages discussed in academia. In particular, the authors observe the architectural languages that have the capability of representing MSA-based systems. Through the use of a detailed query in 4 reliable indexers, a collection of 402 papers were filtered down to a small set of 19 relevant papers. This filtration was done based on a research paper inclusion criteria and a language inclusion criteria. With these papers, a total of 12 architectural languages were investigated for the representation of MSA-based systems.

CCS CONCEPTS

• **Software and its engineering** → *Software design engineering*; *Architecture description languages*; *System modeling languages*; • **Computer systems organization** → *Client-server architectures.*

KEYWORDS

Microservices Architecture, Architectural Language, Domain-specific Language, Service Composition

ACM Reference Format:

Anonymous Author(s). 2022. Architectural Languages in the Microservice Era: A Systematic Mapping Study. In *Proceedings of ACM RACS (RACS'22).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Microservice Architecture (MSA) is a paradigm in the software industry where applications are comprised of small, independent sets of services. Each service carries out a particular business process and performs lightweight communication with other services. This paradigm has evolved from Service-Oriented Architecture (SOA), or traditionally monolithic applications, and provides a number of benefits including fault tolerance, loose coupling, and high scalability [24].

RACS'22, October 3-6, 2022, Japan

ACM ISBN 978-1-4503-8025-6/22/10.
https://doi.org/xx.xxx/xxx_x

Issues arise, however, in MSA-based systems as complexity grows. Maintaining a holistic view of the system becomes increasingly difficult as new requirements and logic are distributed across microservices [23]. One way to address these issues of complexity is through the use of architectural languages (ALs) for microservice systems. An architectural language, as defined by related work from Francesco et. al [2], is any form of expression used for architecture description. Related work from Aksakalli et al. [3] describes how these languages allow complex business processes and interactions between microservices to be described in a way that gives the architect a comprehensive picture of the system. We identify architectural languages as encompassing programming languages and domain-specific languages (DSLs) [11]. Other approaches may exist, but are not covered.

In this paper, a systematic mapping study (SMS) is presented, providing a comprehensive overview of the current architectural languages proposed in academia relating to MSA-based systems. A comparison is made between the languages of their characteristics, features, and research surrounding them. This literature review does not include the architectural languages found in industry without academic research behind them. Findings and conclusions of this study are relevant for industry experts who want a holistic perspective on the languages that exist. This is also helpful for understanding the similarities and differences between these languages, and potential future challenges in architectural languages that may need to be addressed.

The rest of the paper is organized as follows: Section 2 provides background information on work relating to architectural languages. Section 3 describes how the authors aggregated and analyzed papers on architectural languages. Section 4 answers the research questions identified in Section 3. Section 5 discusses the threats to the validity of this study. Section 6 concludes the paper and provides a discussion of future work.

2 BACKGROUND AND RELATED WORK

While previous work does exist that provides an overview of languages for describing system architecture, there is still a lack of research surrounding architectural languages relating specifically to microservice systems. In this section, we describe the related mapping studies, surveys, and literature reviews found, providing a background and context for our research.

Aksakalli et al. [3] conducts a SLR on the current state of deployment of microservices. The paper explains how Amazon, Netflix, eBay, and other companies were found to be using MSA because monolithic applications no longer suited their needs, and MSA proved to have an upgradable and flexible architecture. It explores current methods of deployment and useful tools created for the automatic deployment of microservices. Although there is little mention of architectural languages for microservice systems, the

paper proves that there is widespread growth and adoption of the MSA paradigm in industry.

A mapping study by Francesco et al. [2] evaluates the publication trends and research focus of architecting with microservices. From their selected studies, the authors find a growth in research on the subject in recent years, but note a gap in the amount of research dealing with architectural languages [2]. At one point the study also notes nine architectural languages relating to modeling MSA, but it is not the paper's primary focus. The relevance of this study shows that there is further literature needed with a focus on architectural languages.

Bergmayr et al. [1] provides a systematic literature review on cloud modeling languages. The authors define a cloud modeling language as a domain-specific language in the domain of cloud computing. While the literature review provides an extensive overview of the various cloud modeling languages and their features, it omits microservice orchestration platforms [1]. The study is useful to consider as we investigate architectural languages.

Nikoo et al. [5] presents a related survey on service composition languages. Of the 38 languages found, 14 are explored more deeply in their paper. While some of the described languages pertain to the orchestration of microservices, the study specifically observes languages dealing with service composition [5]. This means other types of languages are also mentioned, and the study does not specifically look at languages for describing microservice systems.

Finally, Malavolta et al. [4] conducts a survey on what the industry needs from architectural languages. The wide array of architectural languages that exist is due to the varying, ever-changing system requirements. The survey conducted interviews of industrial experts to identify the needs from ALs, as well as the useful and unuseful features that exist [4]. The findings of the study reveal a need to reconcile informal architectural languages used in industry with formal ones produced in academia [4]. The study also finds that the most used ALs are produced in industry. This should be kept in mind as we conduct a mapping study of the languages produced through research.

The existing literature previously listed that we observed still shows a gap in discussion on the current state of architectural languages for MSA-based systems. This demonstrates that a mapping study of the languages is a beneficial and worthwhile investigation.

3 MAPPING STUDY METHOD

In this study, we performed a deep investigation of architectural languages and, specifically, those relating to microservices. We followed the guidelines used by Petersen et al. [6] for conducting the systematic mapping study review. Our complete mapping study document can be found below[1]. We began our mapping study by describing key research questions that would help define our overall query. These questions were initially created with a broad perspective and were gradually refined into a final, specific set. The questions we examined in this literature review are as follows:

- RQ1 How do we categorize architectural languages for MSA-based systems and what are these categories?
- RQ2 What are the architectural languages for the specific language categories and what do the languages offer?
- RQ3 What are the challenges for architectural languages and their future directions?

RQ1 addresses how we categorize architectural languages for the purpose of having a comparison framework between the languages. These languages are specifically ones that relate to MSA-based systems. RQ2 addresses the architectural languages found and how they fit into the categories defined in RQ1. RQ3 addresses future challenges and research directions for architectural languages that could be taken based on the findings from our study.

Following the creation of our research questions, we developed a query based on our previously defined areas of interest. This query went through multiple evolutions as we attempted to strengthen the relevance of our results. It was finalized to focus on two main parts: microservices and languages. The microservices part ensures that our query remains in specific relation to microservice architecture. The languages part comprises both ADLs and DSLs, which are commonly referenced in papers to describe an architectural language or languages. Our final query can be seen in Listing 1.

Listing 1: Search Query for Systematic Mapping Study

```
("Architecture-description language*" OR
"language-based" OR "domain specific language"
OR "domain-specific language" OR "DSL"
        OR "ADL") AND microservice*
```

3.1 Search and Filtration Process

Once we finalized our set of research questions, and our query, we performed a search through indexer databases to obtain literature. The databases used for obtaining literature were ACM Digital Library (DL), IEEE Xplore, Springer Link, and Science Direct. By using our final query in these databases, we acquired a total of 402 research papers. The results found per year from this search can be seen in Figure 1. This graph shows an increase in the number of papers published regarding this topic.

The next phase of our literature review was a filtration based on title and abstract. The relevance of a piece of literature was judged based upon a fixed inclusion and exclusion criteria. This process was performed by multiple authors. Our criteria is listed as follows:

Inclusion criteria:

1. Papers investigating architectural language(s)
2. Papers introducing an architectural language
3. Papers expanding upon an existing architectural language
4. Papers providing discussion of architectural language(s)

Exclusion criteria:

1. Not in English
2. Short paper (Less than 4 pages)
3. Non-peer reviewed
4. Duplicate
5. Opinion paper
6. No full text available
7. Published before 2010
8. Literature Reviews

For papers investigating a particular architectural language, or languages, the inclusion of the paper had to meet additional criteria. The first requirement of the paper was that the architectural

[1] https://zenodo.org/record/6867724

Table 1: Search Query Results for Various Index Sites

Indexer	Search Results	Filtered	Snowballed	Total Used
ACM DL	161	3	0	3
Science Direct	135	3	0	3
Springer Link	87	2	1	3
IEEE	19	6	1	7
Other	0	0	3	3
Total	402	14	5	19

language must relate to microservice systems. This means the language was originally developed, or evolved, with the microservice paradigm in mind. The second criterion was that the language must have an explicit name. The reasoning for this was that untitled languages in studies simply demonstrated how their language could satisfy a particular problem, or be used in a particular domain, without attempting to promote industry adoption of their language. This is seen in [21] and [19], where no specific name to their proposed DSL is given. The third criterion was that the language is recently maintained and has substantial research supporting it.

After the exclusion of papers based on title and abstract, we entered the next phase of our literature review which consisted of full reads of papers accepted thus far. During this full read phase, we found additional relevant papers referenced in our filtered papers. These papers were snowballed and used in this study ([9, 13, 14, 22, 25]). Some of these snowballed papers were not published on the indexers we used, and were simply grouped into an "other" indexer category. The specific number of papers that passed the filtration, as well as the the number of papers that were snowballed, are shown in Table 1.

3.2 Data Extraction and Analysis

Once the filtration process was complete, the final phase of mapping and content analysis was entered. During this phase, two authors independently extracted the relevant ALs. Then, the languages were placed into a mind map with branching categories that described each language's application. This extraction was based on codes that we devised for our coding schema. We identify the codes for our coding schema as follows:

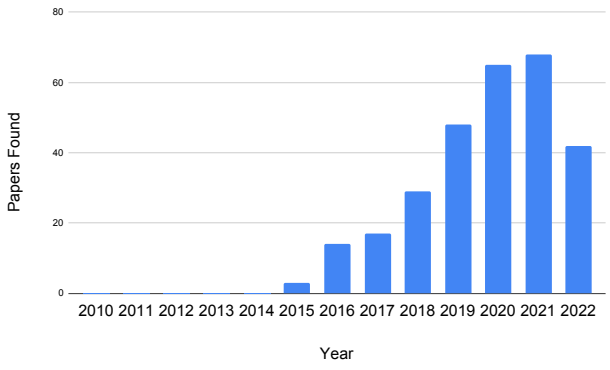

Figure 1: Papers Found Per Year

(1) General information. This includes a language's name, features, particular domain the language is used, and the paradigm the language was developed for.
(2) Challenges. This relates to the problems or hurdles found within a language.
(3) Future directions. This is different from the challenges because this describes future research, rather than simply the problems encountered within the languages.

After the extraction process, we started our data analysis. This analysis assessed how each part of the schema answered our research questions. For RQ1 and RQ2, we looked at the general information we extracted, whether the languages met the inclusion criteria, and whether they had overlapping features or distinct differences. Rather than creating and asking questions to determine the effectiveness of each architectural language in representing an MSA-based system, we chose and compared a set of categories that provided extensive coverage of system concerns amongst the ALs. This was also done for the purpose of answering RQ1. The process of discovering these categories was an iterative one based on observation of overlapping concerns and requirements found amongst the ALs. These categories are listed and further explained in Section 4. In relation to RQ3, we looked at the extracted challenges and directions that were found in the studies. The results of this process are discussed in the following section as well.

4 ANALYSIS RESULTS

Out of 402 papers returned by the search, we found a small number of relevant works. Only 19 papers are considered for the final analysis. The majority of these works focused on introducing one AL as a solution to a particular problem (See [8] and [20]). From our language inclusion criteria described in Section 3, out of a total of 44 languages identified from our results, 12 were included in this literature review. These languages will be further discussed in subsections to follow for the purpose of answering RQ2. The languages included based off the inclusion criteria were Jolie [17], Ballerina [23], Archimate [14], SysML [9], MicroART [12], StratoQL [10], Mu [18], Silvera [22], Medley [8], Partitur [16], BPMN 2.0 [23], and TOSCA [15].

4.1 Categories of Architectural Languages

We outline the following 6 **language categories** that we are concerned with in an AL for the purpose of answering RQ1:

(1) Business Processes: The description of this in an architectural language may be useful for providing a visual notation of the processes [23]. If an architectural language serves the purpose of describing business processes, the language contains higher-level abstractions, rather than lower-level technical details about the system.
(2) Workflows: ALs commonly define services in terms of their logical flow of execution. This flow demonstrates how the services interact with one another (where they send information, and where they recieve information from).
(3) Service Communication: Microservices often communicate and interact with each other, sending data messages back-and-forth. The ability for an AL to describe, and in some instances execute, the data messages that are sent between

microservices may be important for understanding specifically how the microservices interact. This differs from the workflows category in that the flow of execution from one microservice to the next is not described, but instead the potential interchange of data between microservices.

(4) Dynamism: During the evolution of an MSA-based system changes often occur. This feature outlines whether an AL supports the dynamic description of the system. The alternative to this is a static approach to describing the system, that must occur initially at a design phase.
(5) Automatic Generation: Often the user does not want to manually describe their system or write code after describing their system in an AL. Some ALs have an automated solution for either constructing the description of their system, or constructing the code based off of the description. This proves useful as the user only needs to implement their system in one language, and can translate it to another.
(6) Graphical Notation: ALs exist which are paired with a graphical model for visually representing the system. The advantage of this feature is that a visual representation exists alongside the description of the system, which can make understanding the system holistically easier.

In the following subsections, each language category, and the particular languages that fit into them is discussed. This is done for the purpose of answering RQ2. An overall comparison of the languages and their categories can be seen in Table 2.

4.1.1 Business Processes. The languages found that are oriented towards high-level business processes were Archimate, Partitur, and BPMN 2.0 [14, 16, 23]. These languages and their approach to business processes are discussed respectively in subsequent paragraphs.

Archimate is an architecture description language for modeling enterprise architectures. Archimate addresses the category of business processes through its modeling language which incorporates the business, application, and technology layer of systems [14]. Maintained by the Open Group, the language is promoted as an alternative to UML and other modeling approaches for being more coarse-grained in its modeling concepts. Archimate provides language concepts rather than a standard notation due to notations often being stakeholder-specific[2]. Archimate therefore is both extensible, and holistic in providing both high-level business processes, and low-level technical details.

Partitur is a DSL for orchestrating microservices using declarative business processes [16]. The language is introduced as part of the Beethoven platform, which consists of a reference architecture for executing business processes alongside the Partitur language. Partitur uses a set of tasks and event handlers to outline the execution and constraints of the business processes [16]. These tasks and event handlers exist as part of a Partitur workflow. The purpose of this overall is to allow complex business processes distributed across microservices to be easily defined and executed. The authors, however, describe a number of limitations for Partitur. Timeouts and failures must be manually handled, and there is a lack of research around how the language performs on real MSA-based systems [16].

The Business Process Model and Notation (BPMN) Version 2.0 is a domain-specific language used for describing business processes. BPMN has a core set of elements it uses as its structure which is enforced through an XML schema[3]. While BPMN does not quite fit the inclusion criteria of being originally developed or evolving with the MSA paradigm in mind, its applicability towards the paradigm was the basis for its inclusion. Valderas et al. [23] provides an approach for microservice composition in BPMN 2.0 based on the choreography of BPMN fragments, reflecting the capability of BPMN 2.0 as an architectural language applicable for some aspects of MSA-based systems.

4.1.2 Workflows. The languages found that fit into the workflow category, and not the service communication category as well, were Medley, Partitur, and BPMN 2.0. The languages that fit into both categories will be discussed in the subsection dealing with service communication. Archimate is the exception to this, and will be discussed in the following paragraph since it was already discussed in a previous subsection.

Since Archimate, Partitur, and BPMN 2.0 were already introduced, they are briefly discussed in their relation to workflows. Archimate provides the capabilities for modeling both the workflow and communication of the internal services of an MSA-based system through various structural relations between objects, which can then be visualized [14]. Partitur, as previously explained in the business processes category, provides workflow structures for handling business processes [16]. In BPMN, many task components are considered "workflow" tasks and executed in the context of processes[3].

Medley is a domain-specific workflow language proposed by Yahia et al. [8] and further expanded on in [7] for composing microservices. Medley's syntax for describing microservices takes the form of composition structures which hold processes [8]. This allows the expression of both control and data workflows. Further development and research on Medley, however, does not exist since 2016, showing it may be a legacy language [8]. Therefore, updated language alternatives for workflows may be a more suitable choice for developers.

4.1.3 Service Communication. The languages that fit into the category of service communication are MicroART, StratoQL, Mu, and Silvera [10, 13, 18, 22]. Jolie, Ballerina, SysML, and TOSCA also provide service communication alongside workflows [9, 15, 23], and are discussed as well.

MicroART is a MSA recovery tool used to describe and visualize the overall architecture of an MSA-based system [13]. MicroART generates a visualization of a given MSA-based system alongside a generated DSL [12]. The MicroART DSL consists of a meta-model composed of seven metaclasses. The DSL is purposefully minimal, with Interface and Link metaclasses that define lightweight communication protocols, rather than workflows [12]. MicroART, however, is currently a prototype that requires software architect interaction for consistently successful MSA modeling. It also has not been further evolved since being tested on a fictitious benchmark airline service in 2017 [13]. A common pattern emerges that languages which incorporate workflows often do not demonstrate

[2]https://www.opengroup.org/archimate-forum/archimate-overview

[3]https://www.omg.org/spec/BPMN/2.0/

	Business Processes	Workflows	Service Communication	Dynamic Configurability	Automatic Generation	Visualization Component
Programming Languages						
Jolie		✓	✓	✓	✓	
Ballerina		✓	✓	✓		
Domain-Specific Languages						
MicroART			✓	✓	✓	✓
StratoQL			✓	✓		
Silvera			✓	✓	✓	
Mu			✓	✓		
Medley		✓				
Partitur	✓	✓		✓		
BPMN 2.0	✓	✓				✓
Modeling Languages						
Archimate	✓	✓	✓	✓		✓
SysML		✓	✓	✓		✓
TOSCA		✓	✓	✓		

Table 2: Comparison of Language Categories

service communication, with some exception. The reverse of this, as demonstrated with MicroART, is also observed.

StratoQL is a Scala-like, domain-specific language for composing microservices built by the Twitter team [10]. Rather than workflows, StratoQL supports the data interaction between microservices. The language uses a structural type system for writing code that deals with external data and services. StratoQL compiles into an arrows-based, Scala concurrency library named Stitch which provides atomic computations representing calls to services. A corresponding platform called Strato is used with the language to run microservices [10]. This means that StratoQL, when paired with the Strato platform, has not only the capability to describe the system but to execute it as well.

Mu is an embedded DSL for developing microservices in different programming languages, such as Scala and Haskell [18]. Mu has an open type system where the user can decide whether to use types provided by the language, or introduce their own types. Mu provides the functionality for describing services, messages, and requests in its schema. This provides similar service communication capabilities as other languages. The downside of Mu's open type system is that it runs into issues with compilation performance, abstraction mechanisms, and ambiguity [18], which may be an issue for the user depending on what they need from the language.

Silvera is a DSL for MSA-based systems with a compiler that allows the transformation of the Silvera model into code for any programming language [22]. Silvera fits into the category of microservice communication rather than workflows. Silvera defines communication patterns of remote procedure calls and messaging. The syntax of the language then takes the form of services which contain API methods [22]. Silvera is still in the early stages of development, however, and the authors of the language hope to continually improve and develop the language [22].

Jolie is a service-oriented programming language for both simple and complex microservice systems [23]. The general idea of Jolie is that every application written in the language is composed of services and interfaces. The language contains the capability to describe both the execution flow of the system, and the communication between services. This description can then be executed in the programming language as a functioning MSA-based system[4]. The trade off of Jolie being a programming language, however, is that the system must be strictly written in Jolie. While translation into other languages such as Java is possible[4], it is not straightforward, which may be an issue if the user wants their system written in multiple programming languages.

Ballerina is another programming language for integration and orchestration of MSA-based systems [23]. Ballerina has a much larger open-source community than Jolie but offers similar syntax. Ballerina also provides the same workflow and communication capabilities between microservices. The main difference between Ballerina and Jolie is that Ballerina is less strict about its MSA orchestration. In Ballerina, there is also the capability to create non-service functions and executables (i.e. a simple main function) that do not have to be orchestrated under a microservice architecture[5]. Ballerina also does not offer automatic translation from its code to another language, but does provide the capability to embed Java code within Ballerina.

The Systems Modeling Language (SysML) is a modeling language for system of systems (SoS) according to service-oriented architecture (SOA) and MSA principles [9]. System of systems are complex hardware and software systems. SysML models both the workflow of systems composing services, interfaces, and local clouds. SysML depicts workflows through an Arrowhead notation within the language [9]. An arrows-based visualization of the SoS is provided based on this as well. SysML as well can capture the communication between elements [9]. This makes SysML an effective language solution when wanting to capture the complexity of SoS.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) is a domain-specific language for management of cloud services and resources. TOSCA is an OASIS open standard language that provides a blueprint for orchestrating workflows [15]. Alongside this, TOSCA also includes communication between services. There are a variety of approaches to TOSCA, including a YAML-based and XML-based approach [25]. TOSCA does not exist in its own standalone language, but rather is a set of topological guidelines and templates promoted by its creators.

[4]https://www.jolie-lang.org/
[5]https://ballerina.io/

4.1.4 Dynamism. Most of the ALs found provided the capabilities to support dynamism. The exceptions to this were Medley and BPMN 2.0. Medley requires each microservice to be described initially, which means the lack of a dynamic location of microservices may be a potential tradeoff for stakeholders. Partitur is compared with Medley as an alternative that has the capability to dynamically locate new microservices [16]. BPMN exists as a bridge between the business process design, and the process implementation. It is promoted as an initial draft of the business processes given to technical developers. Workflows in BPMN thus must be described before the system is actually implemented[3].

4.1.5 Automatic Generation. The languages that support automatic generation either into a description of their system or code based off the description are Jolie, MicroART, and Silvera. As mentioned when discussing service communication, Jolie does provide the ability to translate the language into Java[4]. While this is possible in the language, it is not a straightforward task. MicroART, by contrast, receives a system from the user, and generates a model and description based off of the system [13]. Silvera, as explained previously, has a compiler that can produce code in any programming language. Currently, the Silvera compiler produces only Java code. This, however, could be extended in the future to support other languages [22].

4.1.6 Graphical Notation. Archimate, SysML, MicroART, and BPMN 2.0 are ALs which also couple a graphical model of the system alongside their descriptions. Archimate's modeling language not only supports the description of a system but also a graphical visualization[2]. SysML as well supports the retrieval of a graphical model based on its modeling language, but in the domain of SoS [9]. BPMN 2.0 supports UML-like diagrams with its notation[3]. The visual models between the languages are notably similar, and should be differentiated by the specific type of application each language is used for.

4.2 Language Target

This section outlines the supported file formats that the found ALs target. This is done as an extension of RQ2 for describing further what these languages offer. The file formats that are supported in each of these languages is displayed in Table 3. This table shows what format each language is targeted towards. Jolie[4] and Ballerina[5] were found to have their own file format (.OL and .BAL). This shows how these languages are targeted towards their own standalone solutions. Silvera, Mu, Medley, Partitur, StratoQL and MicroART[6] are written within other programming languages as an extension [7, 10, 16, 18, 22]. Archimate[2], SysML[7], BPMN 2.0[3], and TOSCA [25] have their language components implemented within XML, XMI, and YAML file formats. These file formats are a good fit for translation across multiple languages.

4.3 Language Applicability

In this section, we discuss and compare which languages are most applicable for the MSA paradigm in order to provide a general understanding of which languages are better suited for modern use. This is for the purpose of further answering RQ2. It should be acknowledged there is no single language that is most applicable over all others due to the different approaches each one takes. While each language is applicable based on specific use cases and requirements, there are certain languages that should be generally considered over others.

Languages	Format
Jolie	OL
Ballerina	BAL
Archimate	XML
SysML	XMI
MicroART	Java
StratoQL	G8
Silvera	PY
Mu	HS
Medley	JS
Partitur	Java
BPMN 2.0	XML
TOSCA	XML, YAML

Table 3: Language File Formats

Due to a lack of further development or adoption in industry, Medley, Partitur, and MicroART seem to be legacy options and therefore less applicable [13, 23]. While Mu was introduced relatively recently, further research since its introduction was not found. Furthermore, the performance and open-type system issues found in Mu likely make the language less applicable when compared to other alternatives [18]. Silvera may be a more applicable option due to its recent introduction and compilation support for other programming languages [22]. StratoQL as well may prove a more applicable option due to its industry use and support by Twitter [10].

BPMN 2.0 seems most applicable towards higher-level business processes, rather than low-level details of MSA-based systems[3]. When comparing Jolie and Ballerina, Jolie is most applicable when looking for a programming language strictly for creating MSA-based systems [17], while Ballerina for a more general-purpose approach[5]. TOSCA has substantial research around it, and seems most applicable when wanting to describe microservices specifically for cloud applications [15].

4.4 Challenges and Future Research

This section outlines the challenges and future directions for architectural languages in order to answer RQ3. There are a number of challenges facing the architectural languages that we found. Due to Jolie and Ballerina being programming languages, they run into hurdles supporting polyglot systems. The ability of an architectural language to support polyglot systems provides advantages in representing a system that encompasses multiple programming languages. While Jolie tries to curb this by providing translation to Java, it does not currently have translation to other languages[4]. Another challenge with architectural languages is the loss of autonomy in language choice. This is seen with StratoQL [10], where the loss of autonomy is described as the most prevalent frustration from the Twitter team. Upward scalability behavior is also a problem that languages such as Medley [7] run into. Providing support

[6]https://github.com/microart/microART-Tool
[7]https://www.omg.org/spec/SysML/

and extension for additional tools also seems to be an overlapping concern [13, 22]. There is significant room for improvement, and additional features are missing throughout these languages.

While this SMS studied a few languages used in industry, such as StratoQL, it mostly focussed on the languages discussed in academia. One area of potential future research is with the architectural languages that are used in industry. These languages do not have as much substantial research in academia. A query could be made on commercial sources such as Stack Overflow for the most commonly discussed architectural languages in industry. These languages may include the ones that related work from Nikoo et al. [5] discusses, such as Netflix Conductor and the Amazon States Language. Additionally, surveying the reception of each particular architectural language amongst multiple teams of developers could prove useful in having an understanding of which languages stand out in the workplace [22]. Finally, with the manual effort required in describing MSA-based systems using ALs, and the existing room for improvement in the automated capabilities of these languages [24], there may exist further research towards that specific capability.

5 THREATS TO VALIDITY

The main threat to the validity of this literature review is the exclusion of relevant research papers. Attempted mitigation was performed against this threat by keeping our research query broad to include as many relevant papers as possible. The exclusion of papers prior to the year 2010 also potentially might have resulted in the removal of older related research. Our manual filtration of papers from the search results based on title, abstract, keywords, and through a full read is also prone to potential human error and the omission of relevant articles. This threat was addressed through multiple reviewers considering each paper and its relevance.

Furthermore, our categorization of ALs may have missed certain system concerns. While this threat was mitigated through the investigation of related works as a foundation, and through the identification of common concerns found across all languages, this is still prone to error. There also may have been relevant languages that were missed due to a lack of research surrounding them, or missed key terms in our search. Due to the wide array of architectural languages that exist, it is also not possible to cover all of them. There are likely ones that are actively used and were not found or included due to these reasons.

6 CONCLUSION

Architectural languages have the ability to provide a holistic view of an MSA-based system[3]. A holistic view can help an architect improve the overall design of a system, catch faults, and detect bottlenecks in the system. In this study, we investigated the architectural languages relating to MSA-based systems and their categories. We outlined 6 language categories: business processes, workflows, service communication, dynamism, automatic generation, and graphical notation. Each of these categories approach describing the architecture of a microservice system differently. It was then discussed which of the found architectural languages fit into each of the categories. The challenges and future directions were then taken based off of the discussed architectural languages. This work, overall, provides an understanding of the current state of architectural languages relating to MSA-based systems.caca

In our future work, we aim to extend an existing architectural language from academia or introduce our own. The user of this language would not have to describe their system manually, but rather could generate the description from their already existing system. This could be similar to the solution MicroART provides [12], and provide a visualization software coupled with the language. This solution would also provide the capability to dynamically describe both workflows and data communication between microservices. This work would be beneficial for users who want to easily retrieve a description of their system and a corresponding a visual model.

REFERENCES

[1] Alexander Bergmayr, Uwe Breitenbücher, Nicolas Ferry, Alessandro Rossini, Arnor Solberg, Manuel Wimmer, Gerti Kappel, and Frank Leymann. 2018. A Systematic Review of Cloud Modeling Languages. *ACM Comput. Surv.* 51, 1, Article 22 (feb 2018), 38 pages. https://doi.org/10.1145/3150227

[2] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2019. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software* 150 (2019), 77–97. https://doi.org/10.1016/j.jss.2019.01.001

[3] Işıl Karabey Aksakalli, Turgay Celik, Ahmet Burak Can, and Bedir Tekinerdogan. 2021. Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software* 180 (2021), 111014. https://doi.org/10.1016/j.jss.2021.111014

[4] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. 2013. What Industry Needs from Architectural Languages: A Survey. *IEEE Transactions on Software Engineering* 39, 6 (2013), 869–891. https://doi.org/10.1109/TSE.2012.74

[5] Mahdi Saeedi Nikoo, Önder Babur, and Mark van den Brand. 2020. *A Survey on Service Composition Languages.* Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3417990.3421402

[6] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

APPENDIX A: THE SELECTED PAPERS

[7] Elyas Ben Hadj Yahia, Inti Gonzalez-Herrera, Anthony Bayle, Yérom-David Bromberg, and Laurent Réveillère. 2016. Towards Scalable Service Composition. In *Proceedings of the Industrial Track of the 17th International Middleware Conference (Middleware Industry '16).* Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/3007646.3007655

[8] Elyas Ben Hadj Yahia, Laurent Réveillère, Yérom-David Bromberg, Raphaël Chevalier, and Alain Cadot. 2016. Medley: An Event-Driven Lightweight Platform for Service Composition. In *Web Engineering*, Alessandro Bozzon, Philippe Cudre-Maroux, and Cesare Pautasso (Eds.). Springer International Publishing, Cham, 3–20.

[9] Jerker Delsing, Géza Kulcsár, and Øystein Haugen. 2022. SysML modeling of service-oriented system-of-systems. *Innovations in Systems and Software Engineering* (09 May 2022). https://doi.org/10.1007/s11334-022-00455-5

[10] Jacob Donham. 2018. A Domain-Specific Language for Microservices. In *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala (Scala 2018).* Association for Computing Machinery, New York, NY, USA, 2–12. https://doi.org/10.1145/3241653.3241654

[11] Haitham A. El-Ghareeb. 2020. 2 - Neutrosophic-based domain-specific languages and rules engine to ensure data sovereignty and consensus achievement in microservices architecture. In *Optimization Theory Based on Neutrosophic and Plithogenic Sets*, Florentin Smarandache and Mohamed Abdel-Basset (Eds.). Academic Press, 21–43. https://doi.org/10.1016/B978-0-12-819670-0.00002-0

[12] Giona Granchelli, Mario Cardarelli, Paolo Di Francesco, Ivano Malavolta, Ludovico Iovino, and Amleto Di Salle. 2017. Towards Recovering the Software Architecture of Microservice-Based Systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW).* 46–53. https://doi.org/10.1109/ICSAW.2017.48

[13] Giona Granchelli, Mario Cardarelli, Paolo Francesco, Ivano Malavolta, Ludovico Iovino, and Amleto Di Salle. 2017. MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems. 298–302. https://doi.org/10.1109/ICSAW.2017.9

[14] Marc Lankhorst, Henderik Proper, and Henk Jonkers. 2010. The Anatomy of the ArchiMate Language. *IJISMD* 1 (01 2010), 1–32. https://doi.org/10.4018/jismd.

2010092301

[15] Paul Lipton, Derek Palma, Matt Rutkowski, and Damian A. Tamburri. 2018. TOSCA Solves Big Problems in the Cloud and Beyond! *IEEE Cloud Computing* 5, 2 (2018), 37–47. https://doi.org/10.1109/MCC.2018.022171666

[16] Davi Monteiro, Paulo Henrique M. Maia, Lincoln S. Rocha, and Nabor C. Mendonça. 2020. Building orchestrated microservice systems using declarative business processes. *Service Oriented Computing and Applications* 14, 4 (01 Dec 2020), 243–268. https://doi.org/10.1007/s11761-020-00300-2

[17] Larisa Safina, Manuel Mazzara, Fabrizio Montesi, and Victor Rivera. 2016. Data-Driven Workflows for Microservices: Genericity in Jolie. In *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*. 430–437. https://doi.org/10.1109/AINA.2016.95

[18] Alejandro Serrano and Flavio Corpa. 2020. Describing Microservices Using Modern Haskell (Experience Report). In *Proceedings of the 13th ACM SIGPLAN International Symposium on Haskell (Haskell 2020)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3406088.3409018

[19] Zheng Song and Eli Tilevich. 2018. PMDC: Programmable Mobile Device Clouds for Convenient and Efficient Service Provisioning. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 202–209. https://doi.org/10.1109/CLOUD.2018.00033

[20] Zheng Song and Eli Tilevich. 2019. Equivalence-Enhanced Microservice Workflow Orchestration to Efficiently Increase Reliability. In *2019 IEEE International Conference on Web Services (ICWS)*. 426–433. https://doi.org/10.1109/ICWS.2019.00076

[21] Gustavo Sousa, Walter Rudametkin, and Laurence Duchien. 2016. Automated Setup of Multi-cloud Environments for Microservices Applications. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. 327–334. https://doi.org/10.1109/CLOUD.2016.0051

[22] Alen Suljkanović, Branko Milosavljević, Vladimir Inđić, and Igor Dejanović. 2022. Developing Microservice-Based Applications Using the Silvera Domain-Specific Language. *Applied Sciences* 12, 13 (2022). https://doi.org/10.3390/app12136679

[23] Pedro Valderas, Victoria Torres, and Vicente Pelechano. 2020. A microservice composition approach based on the choreography of BPMN fragments. *Information and Software Technology* 127 (2020), 106370. https://doi.org/10.1016/j.infsof.2020.106370

[24] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle, and Gastón Márquez. 2021. Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software* 182 (2021), 111061. https://doi.org/10.1016/j.jss.2021.111061

[25] Michael Wurster, Uwe Breitenbücher, Kálmán Képes, Frank Leymann, and Vladimir Yussupov. 2018. Modeling and Automated Deployment of Serverless Applications Using TOSCA. In *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*. 73–80. https://doi.org/10.1109/SOCA.2018.00017