A Qualitative Comparison of Specification Techniques for Microservices Architectures

Jasper van Amerongen
University of Twente
P.O. Box 217, 7500 AE Enschede
The Netherlands
j.vanamerongen-1@student.utwente.nl

ABSTRACT

When designing and developing medium to large scale software systems, architectural decisions are important to get right. One architectural style is the microservices architecture (MSA). Describing and specifying one service or a monolith is well-defined in literature and in practice, however there are doubts on the optimal method of doing so when it comes to a complete MSA. Firstly, this research aims to identify existing techniques for designing and specifying MSAs by performing a systematic mapping study. Secondly, a classification scheme has been applied to perform a qualitative comparison of the found techniques that in turn is used to provide recommendations for business analysts, software architects and developers on which specification technique to use in what scenario.

Keywords

Microservices architecture; MSA; software specification; software architecture.

1. INTRODUCTION

The microservices architectural style is a style of architecting software where a system is split up into relatively small services. These services exclusively maintain their own concern and can scale independently. Microservices architectures (MSAs) have grown rather naturally in practice and have initially been defined in the blog article by Lewis and Fowler [9] that provides a multifaceted overview of MSAs and their applications. In practice, pioneers were mainly tech giants with streaming applications, such as Spotify and Netflix [6, 9], that made the transition to such decoupled architecture.

As a newer approach to Service-Oriented Architectures (SOA), the definition of MSA has not been firmly established at its formal academic conception in Sam Newman's book *Building Microservices* [13], or at least so it has turned out in practice in the last few years. Therefore, Zimmerman has compiled seven tenets to define microservices based on a review of white and grey literature: fine-grained interfaces; business-driven development; following cloud-native design principles; polyglot programming and persistence; lightweight container deployment; decentralised continuous delivery; and DevOps with holistic service monitoring [21]. MSA promotes fine-grained, autonomous services most commonly deployed in cloud environments. In particular, each service runs in an exclusive process (e.g., a dedicated Docker container) and interacts with others through lightweight media [9], like RESTful APIs over HTTP or a shared enterprise message bus (ESB) or message queue (MQ). MSAs can be augmented by concepts such as API gateways, service discovery and circuit breakers [12], which allows it to circumvent what commonly plagues availability, maintainability, scalability and fault tolerance [21].

The inherent fine granularity of MSAs paved the path for the application of Domain-Driven Design (DDD) [5]. This and decentralised governance (i.e., moving towards team's life cycle ownership of software) promises more agile development cycles, benefitting a higher resilience to changing business requirements [9].

Model-Driven Development (MDD) [4] is an approach for developing distributed software architectures, like MSAs. Throughout the development process, MDD utilises models as a means of abstraction. This allows high-level overview and consideration by omitting technical details, analysis through formal denotation practices and speeds up the integration process when code can be generated from the models. MDD plays an important role in the specification techniques covered in this research.

The rest of this document is structured as follows: Section 2 introduces the existing gap in academia and outline the research questions this research aims to answer; Section 3 introduces the chosen methodology and its justification and setup; Section 4 displays and interprets the results found; Section 5 provides recommendations to what specification is appropriate in what situation, based on the previously found results; and this paper concludes with discussing the limitations and threats to validity and suggests future work in Section 6.

2. RESEARCH QUESTIONS

Microservices have shown their benefits as an architectural style for the reasons mentioned before. Looking to move towards MSAs, one challenge is to find and select specification techniques to model such architecture. To contribute to that, we have compiled a list of techniques to specify microservices

TScIT 38, 3 February, 2023, Enschede, The Netherlands

found in literature and compared them by providing guidelines for applying them in various circumstances. In order to do so, we posed the following research question:

What are the leading techniques used in literature and practice to specify a microservices architecture?

This in turn raises the following sub-questions:

1. *Which are the techniques currently used to specify microservices architectures?*
2. *What are the main criteria to compare these techniques?*
3. *What is the result of this comparison?*

When discussing literature in this research, we shall be referring to both grey and white literature, the former being sources authored by field experts, pioneers and researchers that are not necessarily published or peer reviewed. Those may include but are not limited to articles, blog posts and open source software repositories. The justification for the inclusion of grey literature is twofold: 1) grey literature appears to be a common trend in existing white literature and 2) since the topic of MSA is rather young and volatile, as it is initially the professional who may find answers to the questions at hand, as the scientific community has not yet have had enough time to do so with academic rigour.

3. RESEARCH METHODS

This research was divided in three stages, one for each research sub-question, respectively. First, a systematic mapping has been performed to identify existing MSA specification techniques, which gives answer to RQ1. This mapping was set up to be an adaptation of Petersen et al.'s method for systematic mapping studies [14]. Second, the results of this mapping feed into a predefined classification scheme answering RQ2. Third, the identified specification techniques are compared in-depth to answer RQ3.

3.1 Systematic mapping study

We chose to perform a systematic literature mapping for its benefits over narrative, scoping or meta-analysis reviews, which are otherwise commonly employed methods of literature review. The systematic attribute of a systematic mapping allows for less biassed and more inclusive conclusions [8]. Systematic mappings also allow a focus on breadth over depth [8]. Breadth is preferable for navigating the academic landscape for answering RQ1. In contrast, we demonstrate how depth is key for classifying the yielded papers.

Petersen et al. [14] proposed a method for performing a systematic mapping. They have divided this process up in five steps:

1. Definition of Research Questions (Research Scope)
2. Conduct Search for Primary Studies (All Papers)
3. Screening of Papers for Inclusion and Exclusion (Relevant Papers)
4. Keywording of Abstracts (Classification Scheme)
5. Data Extraction and Mapping of Studies (Systematic Map)

In Figure 1, a flow chart describes the systematic mapping pipeline, based on the pipeline proposed by Petersen et al.

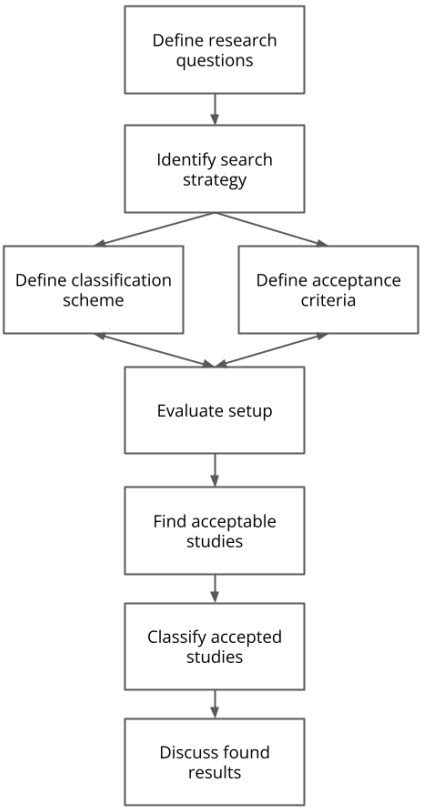

Figure 1: Flow chart of systematic mapping pipeline.

3.2. Search for primary studies

In the pursuit of the primary paper base, the following repositories of academic engineering literature were inquired: IEEEXplore; ACM Digital Library; Springer Link; and Google Scholar.

Two slightly different queries were used for the document title and abstract, respectively. This decision was a result of the iterative evaluation of our systematic mapping setup. As each repository has a different querying interface, specific queries were designed for each, which are shown in appendix A. In general, the search queries used all resemble the following primitives as closely as possible, where "?" denotes zero or one character and "*" denotes zero or more characters.

Document title:

```
Microservice? + (specification | documentation |
model?ing)
```

Abstract:

```
Microservice? + (specification | documentation |
model?ing) + (technique? | method? | approach* |
practice?)
```

For each repository, the complete query result was stored in a spreadsheet along with metadata, such as document title, authors, publication year, journal title, abstract and DOI. The exception to this is Google Scholar, since it utilises an elaborate and inclusive algorithm to determine a search result. Consequently, the size of the query result was around eleven thousand and was deemed too large to be manually surveyed. Ordered by relevance, the first ten pages of results were

considered for this mapping and the rest omitted. The result set form Google Scholar intersected with those from the other repositories and thus were also omitted, preferring the repositories' papers over the duplicates in Google Scholar.

For the sake of comprehension, let the result of this step and the aforementioned substeps be set Π, with each found paper $\pi_i \in \Pi$ and $\pi_i \neq \pi_j$ for $i, j \in \mathbb{N}$.

3.3. Papers Screening

Usually, the result of querying such repositories is a set of papers, a vast portion of which may not necessarily be relevant to answering the research questions [14]. Therefore, the third step in the systematic mapping is distilling down the result by applying inclusion and exclusion criteria to the initial set of papers [14]. This step has been altered in two ways as will be discussed. The acceptance criteria were applied in order that they appear below, i.e., first the inclusion criteria and then the exclusion criteria. They were defined as follows:

Include if

- The title mentions MSA or SOA design techniques;
- The abstract seems to introduce an MSA specific design technique;
- The abstract mentions to build upon existing MSA design techniques yielding a novel one.

Exclude if

- The abstract mentions to research the implementation, applicability, performance or other metric of an existing technique (validation/evaluation research);
- The paper has been published before 2014;
- The paper introduces a technique already found in the mapping and is not the more recent version;
- The paper is not written in English.

These conditions yield a set $\Pi' = \{\pi_i \in \Pi \mid accepted(\pi_i)\}$. Strictly $|\Pi'| \leq |\Pi|$.

Reiteration including skimming

These sets of acceptance criteria turned out to be insufficiently precise. During the manual screening, a notable proportion of potentially relevant papers seemed to be overlooked based on their title and abstract, as the inclusion criteria propose. To compensate, the systematic mapping setup was altered and the screening process was executed again to include papers of which the author deemed them to be equally relevant based on an initial skim of the entire paper. This step was only performed in edge cases, i.e., where the acceptance criteria wrongly excluded the paper. This step updates Π' accordingly.

Second screening layer

After the analysis of title and abstract and a skim of the papers' content, a more thorough screening process was applied. In this step, the papers left after applying the acceptance criteria were read thoroughly to determine whether an MSA specification technique was being proposed. The reason for the second screening layer is to systematically reduce the number of papers that had to be read thoroughly. This was necessary given the provided timeframe in which this research was performed.

Define a set $\Pi'' = \{\pi_i \in \Pi' \mid scanned(\pi_i)\}$. Strictly $|\Pi''| \leq |\Pi'| \leq |\Pi|$.

3.4. Classification Scheme

In order to gain insight into the found and accepted papers, Petersen et al. recommend creating a classification scheme as the next step in their methodology [14]. This should be done by keywording the abstracts by selecting those keywords that reflect the paper's contribution best. The resulting list over all papers can then be used to form categories that represent the context they were formed from. However, as briefly mentioned before, this paper will use a predefined classification scheme. The reason for deviating from Petersen et al.'s method is that the comparison metrics were determined beforehand. Important metrics are completeness, tool support and the most prevalent stakeholders of the specification technique, which were extended with the technique's fabric and whether the technique is defined formally.

Here, completeness is defined as the technique's ability to specify a microservice, a data persistence solution, a fault/error resilience solution, a deployment solution and the microservice(s)' interface(s). The aforementioned submetrics are assessed with boolean values and completeness is assessed as a ratio of these boolean submetrics.

The fabric of a MSA specification technique describes on what medium the technique is used. This can be evaluated with "textual" or "graphical", whether the technique is conveyed through text or the technique is displayed as a diagram or in any other visual manner, respectively.

Tool support is assessed with a boolean value indicating whether or not tools exist or are put forward to adopt the respective specification technique. This does not include trivial tools such as a text editor or pen and paper, but rather custom tools in its totality or as a plugin for, e.g., an IDE.

It was also deemed important to understand which stakeholder benefits the most from each technique. This can be measured with one of $\boldsymbol{\mathcal{P}}(R) \setminus \emptyset$, $R =$ {"Developer", "Maintainer", "Software architect", "Business analyst"}. These roles have been chosen as they were found to properly represent the actors in relevant use cases of MSA specification techniques. E.g., a technique that models MSAs in terms of business requirements appeals more to a business analyst, whilst monitoring fault tolerance appeals more to the system's maintainer.

A technique that has a formal definition is denoted with a boolean value indicating whether the technique has been defined algorithmically or mathematically; by an abstract language through a grammar; or as an metamodelling extension of an existing modelling technique, e.g., an extension of UML.

4. RESULTS

The steps of the previous section were performed in the order they appeared and in line with the method for systematic mappings. The resulting dataset was retrieved between 8 and 12 December 2022.

4.1. Identified specification techniques

Initially, 101 papers have been identified for consideration. After applying the first screening layer, this number was reduced to 40. The second screening lowered this number to 12. As per the second screening layer, each of these papers introduces a novel technique to specify microservice architectures. Table 1 and Figure 2 depict these results. The distribution of repositories of the final set of papers is shown in Figure 3.

Table 2: Distribution of found papers

Repository	Raw	First screening	Second screening
IEEEXplore	15	7	1
ACM DL	6	4	0
Springer Link	32	10	5
Google Scholar	48	19	7
Total	**|Π| = 101**	**|Π'| = 40**	**|Π"| = 12**

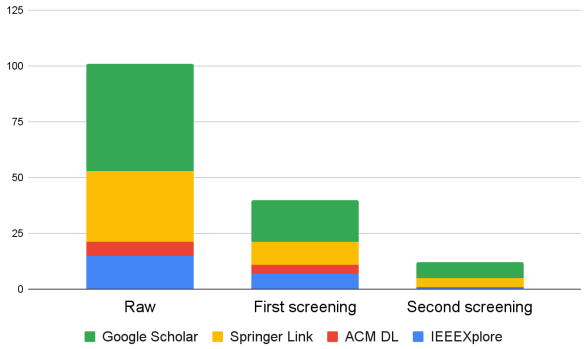

Figure 2: Dataset size after applying two screening layers

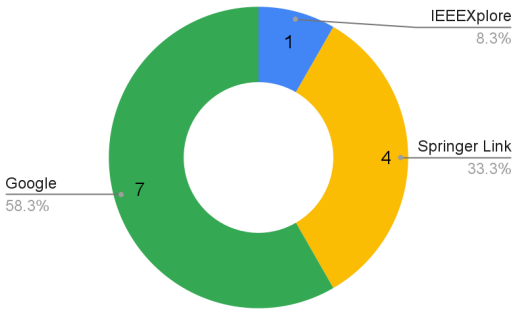

Figure 3: Distribution of repositories in final dataset

4.2. Classified specification techniques

The next step is to catalogue the 12 identified specification techniques according to the classification scheme. Table 3 provides an overview of these techniques alongside their classification attributes. Some cells in the "Input" column are left blank, which indicates that this technique may be used to model MSAs without an initial specification. The subattributes pertaining to completeness are shown in Table 3. The completeness is visualised as a radar plot in Figure 3. Table 5 shows the tool support and the target stakeholder of each technique.

Table 3: Specification techniques' classification metrics

Name	Input	Output	Fabric	Formal definition
MSStack [7]	Proprietary model*[1]	Java	Textual	
Zephyrus2 [1]		Diagram	Graphical	✓
AjiL [16]	AjiML	Java (Spring Cloud)	Graphical	✓
DDMM UML [15]	UML*	DDMM UML	Graphical	✓
Microservice DSL [2]	Monolith	Microservice DSL	Textual	
CBDI-SAE (SOA3) [11]		UML + CBDI-SAE	Graphical	
4SRS-MSLA [17]	UML	UML	Graphical	
Silvera DSL [18]		Silvera	Textual	✓
MicroVision [3]		3D graphs	Graphical	
VxBPMN4MS [19]	BPMN*	VxBPMN4MS	Graphical	
Knowledge Graph [10]	Monolith	Graph	Graphical	✓
BPMN Fragments [20]	Complete BPMN	BPMN fragments	Graphical	

Table 4: Specification techniques' classification metric: Completeness

Name	Completeness					
	Microservice	Persistence	Resilience	Deployment	Interfaces	Ratio
MSStack	✓				✓	0.4
Zephyrus2	✓			✓	✓	0.6
AjiL	✓				✓	0.4
DDMM UML	✓	✓				0.4
Microservice DSL	✓				✓	0.4
CBDI-SAE (SOA3)	✓			✓	✓	0.6
4SRS-MSLA	✓				✓	0.4
Silvera DSL	✓		✓	✓	✓	0.8
MicroVision	✓				✓	0.4
VxBPMN4MS	✓					0.2
Knowledge Graph	✓				✓	0.4
BPMN Fragments	✓					0.2

[1] The paper in which this technique is presented puts forward a proprietary modelling technique that serves an input to the actual tool that is MSStack. It has therefore been determined that this modelling technique is the input if MSStack is considered on its own. However, this modelling technique is part of the workflow of the presenting paper and therefore the argument stands to consider this cell empty.
* One might adapt this input or start from a blank slate to directly obtain the output model.

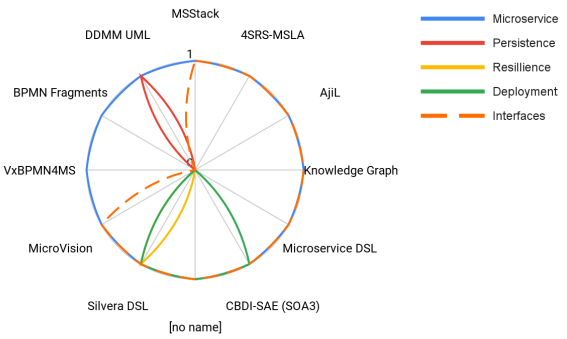

Figure 4: Radar plot of specification techniques' classification metric: Completeness

Table 5: Specification techniques' classification metric: Management

Name	Management	
	Tool support	Stakeholder
MSStack		Business analyst, Developer, Architect
Zephyrus2		Maintainer
AjiL	✓	Developer, Architect
DDMM UML	✓	Developer, Architect
Microservice DSL	✓	Developer
CBDI-SAE (SOA3)		Developer, Architect
4SRS-MSLA	✓	Business analyst, Developer
Silvera DSL	✓	Developer
MicroVision	✓	Developer, Architect
VxBPMN4MS	✓	Business analyst, Developer, Architect
Knowledge Graph		Developer, Architect
BPMN Fragments	✓	Business analyst, Developer, Architect

5. RECOMMENDATIONS

In addition to the systematic mapping provided above, this paper provides the following set of recommendations for which technique to employ in a given situation. These recommendations are based on each technique's position in our classification scheme and their attributes and this is implied in each recommendation's justification. Any further justification is based on contributions of the papers in which the techniques are presented.

5.1. Greenfield

Greenfield software systems are those that start off in a completely new environment and no legacy code is present. In the context of this paper, this can mean one of two scenarios: no formal specification yet exists and will thus have to be constructed; or the software to be developed has been modelled in some way in compliance with high-level business requirements as specified by existing business modelling languages, such as UML or BPMN.

With prior business requirements & processes

MSStack is a framework for developing an MSA based on Java with "built-in logging, monitoring, load balancing, and scaling capabilities" [7]. Important for this paper is the proprietary business model (see footnote 1) that will be the input for the native model-to-code conversion module. This model is inspired by BPMN, but it additionally also supports modelling relevant data entities to enable the modelling of the overall business domain [7]. The complete suite that comprises the framework allows seamless cooperation between business analysts, software architects and developers.

Domain-driven MSA modelling (DDMM) UML [15] is a UML profile which extends UML metaclasses with stereotypes. Its respective paper fills the gap between DDD and UML modelling, which allows this specification technique to inherit from both. DDD has shown to be a crucial consideration when architecting MSAs [15].

VxBPMN4MS has been introduced to allow its users to model business processes whilst taking variability into account by extending the BPMN language [19]. Like MSStack, the respective paper also puts forward an automation framework for establishing an MSA and thus inherits those same properties relevant to the context of this paper as MSStack

Valderas et al. [20] put forward a decomposition technique to fragment BPMN models. The resulting BPMN fragments each represent a microservice. This method allows a big-picture overview through traditional BPMN, whilst the fragmented version allows modelling the choreographic composition. This approach is said to aid further analysis when making engineering decisions [20].

All of the four specification techniques mentioned above allow modelling an MSA after the input from the business side and should thus be used in such situations. DDMM UML, VxBPMN4MS and BPMN fragments are especially recommended if it is expected that the specifications will be further developed in their respective modelling languages. Additionally, in case a technique covering the entire design-to-deployment pipeline is sought, MSStack should be the chosen candidate.

Without prior specification

Silvera [18] is a Domain-Specific Language (DSL) and compiler for modelling and developing MSAs. It is based on the principles of Model-Driven Development (MDD) and is said to be easy to be used by both domain experts and beginners [18]. Some highlighted features include a retargetable plugin-based compiler, an automatic documentation generator and MSA tailored metrics for architecture evaluation.

It is because of Silvera's ability to model and develop MSAs and nullish input that this specification technique is ideal for previously unspecified greenfield MSAs. MSStack and DDMM UML tick the same boxes and can therefore also be considered for this type of greenfield MSA. In addition, Silvera's rich collection of features and compile targets, completeness across the entire stack and proven ease of use makes it the recommendation for any greenfield MSA.

5.2. Brownfield

In the context of software development, brownfield development entails the architecting and developing of software systems alongside, based upon or in close proximity to legacy code. In this section, we cover the scenario where a monolithic application is sought to be transformed into an MSA. Microservice DSL, 4SRS-MSLA and a knowledge graph are each techniques to model this change and microservices during and after such changes.

Bucchiarone et al. present a model-driven approach for migrating to an MSA [2]. This process is composed of two components with a respective DSL. One of them is Microservice DSL and is the extracted specification technique in this paper. Microservice DSL relies on Jolie, but with added concepts as typed interfaces for message exchange and input and output ports. The DSL is verbose in its definition of MSAs, which aids readability.

4SRS-MSLA is a four-step rule set (4SRS) for deriving a Microservice-oriented Logical Architecture (MSLA) from a monolith, that provides a logical view on the behaviour of and relationships between microservices [17]. This is modelled through UML use case diagrams. This specification technique models MSAs as a collection of bounded UML components with specified rules for pruning suboptimal interactions. It should be noted that this technique need not necessarily be a result from the 4SRS and thus this technique can be used for greenfield MSLAs as well.

A technique to decompose a monolithic application based on graph theory is the one presented by Li et al. [10]. A monolithic application is represented with a graph, where each node represents a data entity, module, function or resource and weighted edges represent relationships between them. Then communities are detected by means of the Louvain algorithm which can be aggregated into microservices.

For the extraction of an MSA from a monolith the recommendation is to employ the 4SRS-MSLA or the knowledge graph technique, as they provide clear guidelines on how to do so and the possibility to model not only the initial and final states of the extraction, but also the intermediates. Additionally, such graph-like representation of both is expected to allow deeper analysis of each step by applying other mathematical tools. If the microservice extraction is desired to be executed by means of DSLs, the Microservice DSL (and Deployment DSL from the same paper) should be chosen.

5.3. High-level spatial overview

Like Silvera, AjiL [16] is a specification technique to enable MDD of MSAs. In contrast, AjiL is a graphical approach. With the complementary Eclipse plugin, two types of diagrams can be drawn up to be generated into code. The overview diagram is suited to model an MSA's topology, whereas the detailed diagram allows specifying components, such as interfaces and data entities, for each microservice. The simplicity of AjiL allowed its researchers to successfully apply it in educational settings and in designing MSAs with a small number of services [16]. This is a double-edged sword, however, as they also observed that the simplistic and human readable representation may lack expressiveness and precision in larger applications [16]. The recommendation extends their findings that AjiL is best applied in smaller and/or less technical MSAs, where spatial overview is regarded more than precise details.

A specification technique that transcends the two-dimensional plane is MicroVision [3], using Augmented Reality (AR) to represent an MSA in 3D space. In order to construct such visualisation, each microservice's abstract syntax tree is analysed and a call tree is constructed. API endpoints are determined by top-level functions together with their annotations. The call tree is stored in a graph database, which is then reconstructed in AR. Like AjiL, MicroVision presents two ways: an overview and API view. Both are represented as a graph and in the latter specifically, connections are highlighted and the selected API's interface displayed in a table. The added dimension in this technique and the ability to physically move through the representation makes it appealing to relatively tightly-coupled MSAs when their interactions would otherwise get obscured in other techniques.

6. EVALUATION

This section elaborates on the limitations and validity related to this research. Limitations are mostly in the form of a short timeframe and validity is threatened by a small sample size or not sufficiently detailed classification scheme.

6.1. Limitations

The limitations of this paper were exclusively rooted in a stern deadline. This paper was proposed, executed, drafted and finalised in ten weeks, with six reserved for execution and drafting. A more generous timeframe would have allowed this paper to improve on the following points:

- **Larger dataset**
  An increase in time would have allowed for more repositories of scientific papers to be queried and with possibly less exclusive queries. Also, more papers could have been collected from Google Scholar.
- **More extensive comparison metrics**
  Another limitation is the relatively shallow comparison that was performed. To gain a deeper understanding of the found techniques more elaborate comparison metrics must be used. A deeper understanding is expected to then lead to more representable recommendations.
- **Real-world tests**
  Performing real-world tests with these techniques by setting up a case study and attempting to specify such microservice architecture by the hand of each found technique would likely further aid the validity of the

comparisons and recommendations. Moreover, this would allow the classification scheme to be extended with performance metrics, e.g., temporal, compute, latency, throughput performance. Real-world tests could also be surveyed with field experts, such that developer experience (DX) metrics can be employed to the classification scheme.

- **Analysis of related work sections**
  In the process of analysing the papers that were found in the mapping, not earlier found techniques were mentioned in some of the related works sections. These were not found in the mapping, however. Extracting techniques from the related works sections was not part of the systematic mapping and these techniques were thus ignored. This means that we might have missed relevant papers. Due to time limitations, the mapping study was not performed again.
- **More comprehensive textual coverage**
  Our comparison was based on their characteristics in the classification scheme and other properties that were outlined in the papers presenting the techniques. The latter has been summarised in the respective section of that technique. This has been kept brief, however. In future iterations, it would be better for the sake of completeness to elaborate more on each technique. This will allow a deeper comparison and with that a recommendation that is more complete.

6.2. Threats to validity

It stands to reason that validity might suffer from the fact that the classification scheme has been designed for specification techniques of MSAs specifically and thus with no regard to any possible standards for classifying specification techniques that might be found in literature. The classification scheme that is used in this paper is not insusceptible for bias and classifying based on different metrics is likely to alter our concluding recommendations.

What might also impair the validity of the posed recommendations is a shallow classification scheme. More elaborate classification schemes based on multivocal comparison metrics could be defined and applied. This extension should increase the validity of conclusions drawn from it.

7. CONCLUSION

In this paper, a systematic mapping study has been designed and performed to map the academic landscape of MSA specification techniques. Twelve papers that put forward a specification technique have been found through this method. These twelve techniques have been evaluated by means of a predetermined classification scheme. This evaluation was used to form recommendations in three scenarios. In the case of a greenfield environment Silvera [18] is the recommended specification technique. In brownfield environments we recommend the use of the 4SRS-MSLA [17] to specify and decompose a monolithic software system. Alternatively, a knowledge graph can be formed by means of the technique in [10]. When it is desired to create a high-level overview, AjiL [16] is recommended for MSAs with a relatively little number of microservices. MicroVision [3] is recommended for MSAs with a relatively high number of microservices or where microservices are tightly coupled.

7.1. Future work

In the process of forming the recommendations, we noticed that VxBPMN4MS and BPMN fragments both use the BPMN modelling language and DDMM UML and 4SRS-MSLA both use UML. This inspires questions such as: Can techniques that are built on the same modelling language interoperate in one model? Does this allow the result to benefit from both models' strength? Or rather the opposite? Future work might include answering these questions.

As with any literature review, this is a snapshot in time. Thus future work might also include performing a similar systematic mapping study again. A repetition of this study allows the authors to take the limitations of the current execution into account.

REFERENCES

[1] Bravetti, M., Giallorenzo, S., Mauro, J., Talevi, I., & Zavattaro, G. (2019). A Formal Approach to Microservice Architecture Deployment. Microservices, 183–208. https://doi.org/10.1007/978-3-030-31646-4_8

[2] Bucchiarone, A., Soysal, K., & Guidi, C. (2020). A Model-Driven Approach Towards Automatic Migration to Microservices. Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, 15–36. https://doi.org/10.1007/978-3-030-39306-9_2

[3] Cerny, T., Abdelfattah, A. S., Bushong, V., Al Maruf, A., & Taibi, D. (2022). Microvision: Static analysis-based approach to visualizing microservices in augmented reality. 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). https://doi.org/10.1109/sose55356.2022.00012

[4] Combemale, B., France, R. B., Jézéquel, J. M., Rumpe, B., Steel, J., & Vojtisek, D. (2017). Engineering Modeling Languages: Turning Domain Knowledge Into Tools. Taylor & Francis.

[5] Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Professional.

[6] Goldsmith, K. & GOTO Conferences. (2015, December 23). Microservices at Spotify [Video]. YouTube. Retrieved January 22, 2023, from https://www.youtube.com/watch?v=7LGPeBgNFuU

[7] Jayawardana, Y., Fernando, R., Jayawardena, G., Weerasooriya, D., & Perera, I. (2018). A Full Stack Microservices Framework with Business Modelling. 2018 18th International Conference on Advances in ICT for Emerging Regions (ICTer). https://doi.org/10.1109/icter.2018.8615473

[8] Kitchenham, B.& Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University.

[9] Lewis, J., & Fowler, M. (2014, March 25). Microservices. martinfowler.com. https://martinfowler.com/articles/microservices.html

[10] Li, Z., Shang, C., Wu, J., & Li, Y. (2022). Microservice extraction based on knowledge graph from monolithic applications. Information and Software Technology, 150, 106992. https://doi.org/10.1016/j.infsof.2022.106992

[11] Ma, Z., Liu, J., & He, X. (2018). An Approach to Modeling Microservice Solutions. Information Science and Applications 2018, 533–542. https://doi.org/10.1007/978-981-13-1056-0_53

[12] Montesi, F. & Weber, J. (2016). Circuit Breakers, Discovery, and API Gateways in Microservices. arXiv. https://doi.org/10.48550/arxiv.1609.05830

[13] Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems (2nd ed.). O'Reilly Media.

[14] Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. Electronic Workshops in Computing. https://doi.org/10.14236/ewic/ease2008.8

[15] Rademacher, F., Sachweh, S., & Zündorf, A. (2018). Towards a UML Profile for Domain-Driven Design of Microservice Architectures. Software Engineering and Formal Methods, 230–245. https://doi.org/10.1007/978-3-319-74781-1_17

[16] Rademacher, F., Sorgalla, J., Wizenty, P., Sachweh, S., & Zündorf, A. (2019). Graphical and Textual Model-Driven Microservice Development. Microservices, 147–179. https://doi.org/10.1007/978-3-030-31646-4_7

[17] Santos, N., Salgado, C. E., Morais, F., Melo, M., Silva, S., Martins, R., Pereira, M., Rodrigues, H., Machado, R. J., Ferreira, N., & Pereira, M. (2019). A logical architecture design method for microservices architectures. Proceedings of the 13th European Conference on Software Architecture - Volume 2. https://doi.org/10.1145/3344948.3344991

[18] Suljkanović, A., Milosavljević, B., Inđić, V., & Dejanović, I. (2022). Developing Microservice-Based Applications Using the Silvera Domain-Specific Language. Applied Sciences, 12(13), 6679. https://doi.org/10.3390/app12136679

[19] Sun, C. A., Wang, J., Liu, Z., & Han, Y. (2021). A Variability-Enabling and Model-Driven Approach to Adaptive Microservice-based Systems. 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC). https://doi.org/10.1109/compsac51774.2021.00130

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

[21] Zimmermann, O. (2016). Microservices tenets. Computer Science - Research and Development, 32(3–4), 301–310. https://doi.org/10.1007/s00450-016-0337-0

APPENDIX A.

IEEEXplore:

```
("Document Title":"microservice?") AND ("Document Title":"design" OR "Document Title":"specification" OR "Document Title":"documentation" OR "Document Title":"model?ing") AND ("Abstract":"microservice?" OR "Abstract":"MSA") AND ("Abstract":"design" OR "Abstract":"specification" OR "Abstract":"documentation" OR "Abstract":"model?ing") AND ("Abstract":"technique?" OR "Abstract":"method?" OR "Abstract":"approach?" OR "Abstract":"practice?")
```

ACM Digital Library:

```
"microservice?" AND ("design" OR "specification" OR "documentation" OR "model?ing")
```

```
"microservice?" AND ("design" OR "specification" OR "documentation" OR "model?ing") AND ("technique?" OR "method?" OR "approach*" OR "practice?")
```

Springer Link:

```
microservice? (specification OR documentation OR model) (technique OR method OR approach OR practice)
```

```
microservice?
```

Google Scholar:

```
microservice (specification OR documentation OR model) (technique OR method OR approach OR practice)
```