Tools Reconstructing Microservice Architecture: A Systematic Mapping Study

Alexander Bakhtin[1], Xiaozhou Li[1], Jacopo Soldani[2], Antonio Brogi[2], Tomas Cerny[3], and Davide Taibi[1,4]

[1] University of Oulu, Oulu, Finland,
[2] University of Pisa, Pisa, Italy
[3] University of Arizona, Tucson, Arizona, USA
[4] Tampere University, Tampere, Finland

Abstract. Various tools have been developed to reconstruct the microservice system architecture. Some of the main reasons to build yet another architectural reconstruction tool are the lack of features to satisfy the current needs or the fact that researchers are often unaware of the existing tools. To shed light on the available tools, we performed a review of the literature in the form of a systematic mapping study to identify the different architectural reconstriction tools adopted in research works, classifying their purpose, input, and output. This paper compares 37 tools. Out of these, 19 are based on static analysis, 10 on dynamic, and 8 using a combination of them. The study shows a significant overlap among tools, with several unmaintained, abandoned, or unavailable. This work will help researchers identify the architectural reconstruction tools that fit their purposes rather than developing another similar tool. This work includes an online appendix [1].

Key words: Microservice, Software Architecture, Architectural Reconstruction

1 Introduction

Microservices bring significant benefits to stakeholders involved in software development and deployment. Development teams work in smaller, autonomous units focused on specific services, which enables decentralization. However, there come times when we need to see the system as a whole to make informed decisions on maintenance and evolution. The problem with the decentralization that allows teams to work more independently is that teams understand microservice bounded context but do not see beyond. With access to a system-centered view, they could better strategize for optimization, patches, and new features, not introducing changes that could deteriorate system design and its operability [2].

We typically look into the system architecture to understand the system as a whole. However, there is no guarantee that the planned architecture matches the actual architecture since the system is developed decentrally and constantly evolves without the means to assess whether the architecture maintains the prescribed format. For example, Baabad et al. [3] synthesize work by Taylor et al.

[4] and Perry and Wolf [5] and conclude that they *described the architectural degradation as a process of the persistent inconsistency between the descriptive software architecture as implemented and the prescriptive software architecture as intended*. Thus, to understand systems, we typically perform software architecture reconstruction [6]. However, the challenge with microservices and decentralized teams is determining the system-centered view of separately designed parts, possibly involving different codebases and separate issue-tracking systems. While system monitoring offers certain means to discover service dependencies, these have limits to the extent of uncovered detail and completeness [2, 7].

This work aims to identify available tools for reconstructing the system architecture from microservice systems. For this reason, we perform a Systematic Mapping Study [8] (SMS) identifying 37 tools, categorizing them based on common goals, supported platforms, benefits, and outputs behind such reconstructions, along with the common inputs to such tools to provide the community with a comprehensive overview to exiting tools to apply or extend rather than reinventing the wheel.

This paper is structured as follows: Section 2 discusses the related works, Section 3 describes the adopted SMS method, Section 4 presents the results and describes what information we gather about the discovered tools, Section 5 goes into further discussion of results, and Section 6 concludes the paper. This paper also has an online appendix [1] that provides tables with detailed information about the discovered tools as well as a list of all papers that came from the SMS.

2 Related Works

Various secondary studies try to organize the ever-growing body of research on microservices. The first attempts, to our knowledge, were [9] and [10]. In [9], authors considered what was proposed by Academic literature concerning microservices up until that point (2016), focusing on proposed views and metrics but not on tools providing them. In [10], authors conduct an SMS to identify different types of microservice architectures as well as tools enabling to create projects with microservice architectures, but not tools extracting them.

Another notable attempt in this direction is [11], which analyzed the state-of-the-art on Microservice Architectures (MSAs). The goal of [11] was indeed to report on the evolution of software architectures into microservices and to describe open research challenges. Other examples are [12, 13, 14]: [12] presents the results of a Grey Literature Review aimed at analyzing the practitioners' view on the "pains and gains" of microservices; [13, 14] instead elicit the architectural/security smells for MSAs and the refactorings, allowing to resolve them by conducting Multivocal Literature Reviews. However, the four studies mentioned above all differ from our review in the aim of the study and in the method exploited to pursue that aim. Similar considerations apply to [15, 16], which run Systematic Literature Reviews to pursue different aims than ours. [15] actually focuses on the deployment/communication patterns used in MSAs, while [16] focuses on failure detection and root cause analysis in MSAs.

Other secondary studies worth mentioning are the Systematic Mapping Studies in [17, 18, 19] and the Rapid Review in [20]. [17] and [20] both consider the reconstruction of MSAs as part of the broader scopes of analyzing and reasoning on MSAs. [18] and [19] instead classify the existing techniques for visualizing antipatterns in MSAs and service interactions in running MSAs, respectively. Despite the fact that [17, 18, 20] touch the topic of MSA reconstruction and visualization, they are not eliciting nor classifying the existing tools for running such tasks, which is instead the aim and scope of this study.

The work by Cerny et al. [6] is perhaps the closest study to ours. The Systematic Literature Review in [6] elicits and classifies the existing techniques for reconstructing the architecture of existing MSAs by distinguishing between static and dynamic reconstruction techniques and by also commenting on how reconstructed MSAs can be visualized. The focus of [6] is, however, on the *techniques* for reconstructing and visualizing architectures, assuming that they are already designed as MSAs, and the work only mentions whether/how they have been implemented.

Similar considerations apply when relating our work to existing primary studies for reconstructing and visualizing MSAs, e.g., [21, 22, 23, 24]. The existing primary studies indeed typically focus on proposing *techniques* for reconstructing and visualizing architectures, which are sometimes released also through prototypical implementations.

The focus of our study is, therefore, different: we indeed review the existing *tools* for reconstructing MSAs, including both the migration of monoliths to microservices, the reconstruction of existing MSAs, and the possibility to visualize the obtained results.

3 Methods

This section describes the method we applied to identify and classify the existing tools reconstructing MSAs.

3.1 Research questions

Our goal is to catalog existing tools that have been introduced to the community with a scientifically published work. We, therefore, formulated the following research questions:

RQ$_1$ *What tools for microservice reconstruction have been developed?*
RQ$_2$ *What languages/platforms are currently supported by the tools?*
RQ$_3$ *What is the purpose of the reconstruction?*
RQ$_4$ *What is the input/output of the tools?*

In order to answer our RQs, we adopt a Systematic Mapping Study of the literature according to [8]. We also perform snowballing on the found papers according to guidelines by Wohlin [25]. Both original and snowballed papers are filtered with the use of Inclusion and Exclusion criteria. We then extract the tools from the selected papers.

3.2 The Search Process

To answer our research questions, we searched for scientific literature introducing tools for reconstructing MSAs. Following the guidelines provided in [8], we identified the search string by structuring it guided by our research questions. More precisely, we defined the search string based on the terms characterizing our research questions, picking keywords in order to cover the four main aspects of our research question. As a result, we obtained the following search string:

```
(Microservice* OR Micro-service* OR "micro-service*") AND Architect*
AND (Reconstr* OR Mining OR Reverse engineering OR Recover* OR Extract*
                                OR Discover*)
AND (Tool* OR Prototype OR Implementation OR GitHub OR Proof of concept
                        OR POC OR Proof-of-concept)
```

(where "*" matches lexically related terms, e.g., plurals and conjugations). In the search string, the first OR-group accounts for different spellings of the term "Microservice", and the third provides additional synonyms to the term "Reconstruction". The fourth OR group was applied to search in-text if permitted by the database's search syntax/filters.

The search string was used to search for literature on the following scientific databases by converting it to the appropriate syntax: Scopus,[1], IEEEXplore[2], ACM Digital Library,[3] and the citation database Web of Science[4]. Initially, only the Scopus search was performed on **7th of February 2023**. The decision to include other databases occurred after a team discussion of initial results, and the queries were performed on **23rd of February 2023**.

The counts of obtained papers are: Scopus - 369, IEEE - 71, ACM - 28, Web of Science - 114; Total excluding duplicates is 387.

3.3 Selection of papers

After compiling the initial list of 387 papers found by the query, we proceeded with a read of the title and abstract of each paper to determine if it was in the scope of our research and worthy to be investigated for tools. We used the following inclusion and exclusion criteria to guide the paper selection process:

- **Inclusion criteria**
  - Mentions a tool in the context of microservice reconstruction
- **Exclusion criteria**
  - Material not in English
  - Out of topic - terms used with different meanings
  - Different aspects of microservice reconstruction (not dealing with tools)

[1] The Scopus database: https://www.scopus.com.
[2] The IEEEXplore database: https://ieeexplore.ieee.org/.
[3] The ACM Digital Library: https://dl.acm.org.
[4] Web of Science: https://www.webofscience.com/wos/woscc/basic-search

The inclusion of the paper was determined by two authors separately (from the first three authors of the paper). In case of disagreements, a third author (from the last three authors) resolved the disagreement. After looking at some initial pool of results, we decided to be as inclusive as possible towards papers mentioning some kind of tool in order to create a more comprehensive catalog of proposed tools in the context of MSA reconstruction.

In particular, during piloting, we noticed that three distinct areas use the term 'Microservice Reconstruction' with different meanings:

- Microservice Architecture Reconstruction, i.e., construction of a 'map' of Microservice systems, showing how different microservices connect to each other. *Main interest of this study.*
- Monolith to Microservice Migration, i.e., clustering of methods/classes of monolithic applications into distinct microservices. These papers frequently say, "We *reconstruct* the Microservice architecture from a Monolithic system," even though the correct word to use here would be *construct*, since a novel architecture is created.
- Microservice recovery, i.e., redeploying microservices that crashed due to an error. These papers say that "Microservices are *reconstructed* from a failed state," using *reconstruct* as a synonym for *recover*.

After observing these results during piloting, we decided to accept papers from the first and second contexts but reject papers from the third context. The decision to include, during this stage, Monolith to Microservice migration tools is explained by the hope that some of these tools might be 'tricked' to accept a Microservice system as input and get its real architecture as output.

The number of papers selected from 387 in our case was 81.

3.4 Tool extraction/Snowballing

After selecting the papers in the previous step, we proceeded with a full read in order to extract the existing tools for MSA reconstruction. We extracted the tools that were directly employed by each paper (e.g., the paper introduces the tool or the tool is studied/applied in the paper), as well as any other tools with similar functionalities that are mentioned in the selected papers' sections (e.g., in their introduction, background, or related work discussion). As such, this process was combined with backward and forward snowballing on the papers, namely finding additional resources by following citations [25]. Backward snowballing was performed on citations in the selected papers, as well as forward snowballing using Google Scholar to find papers that cite the selected papers. Since the process was combined with extracting the tools, we could quickly see that despite finding additional papers using snowballing, the tools they mention are the same tools we find from the originally selected papers.

For each paper, one person extracted the tools. However, if he reported that no tools are found, another author stepped in to read the paper and confirm that. In the online appendix [1], we report for each tool all the papers among the selected papers that cite the tool.

In our case, we added 14 papers by snowballing and extracted a total of 37 tools.

3.5 Tool coding

For each tool, we collect information on the development activity, license, supported languages, platforms or frameworks, input, and output. We also assessed the architectural recovery method and the existence of visualizations that make information about certain system aspects accessible to users. One author extracted each piece of information in a shared spreadsheet. Then, at least three authors collaboratively classified them using a collective coding method. Incongruences were discussed until disagreements were resolved.

The final coding process led to the collection and classification of the information included in the following taxonomy:

- `Tool name`
- The `reference` of the paper introducing the tool
- All selected papers that `cite` the tool
- `Tool repository` information:
  - `Availability`[5]
  - Indicated `license`
  - `Last update/commit date` (as of 23rd of June 2023)
  - `Total amount of commits`
  - `Number of stars`
  - `Number of forks`
  - `Number of contributors`
- `Supported language/platform/framework`
- `Input`:
  - `Input type`:
    - `Source code (Source)` - original source code in plaintext form; can also refer in particular to git repositories if git history is studied
    - `Model-generatable from source (Model-GFS)` - Some intermediate representation of code/repository/infrastructure/etc. that can be automatically generated by existing tools as it adheres to a standardized format
    - `Model-manual custom format (Model-MCF)` - Some intermediate representation of code/repository/infrastructure/etc. that needs to be manually generated (or a custom generator written) since authors define the format themselves

[5] We do not provide links to save space in the table. If the repository is indicated as available, its name in Table 1 is a hyperlink in the electronic version of the paper. Additionally, it can be found in the introductory paper of the tool from Table 4 of the online appendix [1].

    - `Traces` - Special type of logs, usually implemented using OpenTracing standart[6]
    - `Deployment files` - Files necessary for Docker/Kubernetes deployment, such as `Dockerfile`, `docker-compose.yml`, Kubernetes manifest
  - `Input format` - Free-form clarification of the particular case
- `Output`
  - `Output type`:
    - `Smells, patterns, anti-patterns`
    - `Architectural views [22, 26]` - target system aspects to describe, i.e., service view (describing the service models that specify microservices, interfaces, and endpoints), domain view (describing the entity objects of the system as well as the data source connections of those objects.), operational view (describing service deployment and infrastructure, such as containerization, service discovery, and monitoring), etc.
    - `Health metrics` - For monitoring tools, data they provide that can be used to infer the status and health of the project
    - `Tests`
    - `Refactorings, violations`
  - `Output sub-category` - Free-form clarification of a particular case, common values are:
    - `Service Dependency Graph (SDG)` - a graph that shows which services call one another
    - `Class to Microservice mapping (C2M)` - for monolith to microservice migration tools, the proposed grouping/refactoring of existing classes/methods into microservices
  - `Output format` - The particular format (JSON, CSV, microTOSCA, etc) that the tool produces
  - `Recovery method` - One of the following:
    - `Static` - source code/repository is analyzed without building and running the project
    - `Dynamic` - project is run, and runtime data (logs, metrics) are collected to perform the analysis
    - `Hybrid` - data from both stages are used
  - `Tool aims` - Overall purpose of the tool, common values are:
    - `Microservice Reconstruction (MR)` - mapping out the SDG of a microservice system
    - `Monolith to Microservice Migration (M2M)` - proposal of grouping methods/classes of a monolithic system into microservices
    - `Vulnerability detection (VD)`
    - `Smell detection (SD)`
    - `Pattern detection (P)`

[6] A couple of tools use actual logs and not traces, but we decided not to introduce another category for this case

    - `Monitoring (MO)`
  - `Visualization` - whether the tool produces some kind of visualization

4 Results

A total of 81 papers are selected by a full read from 387 obtained by search, to which an additional 14 are added by snowballing; a total of 37 tools are identified. The summary of results is reported in Table 1. Detailed information about the tools is provided in the online appendix [1]. We also provide the introductory paper as well as all referencing papers for each tool in Table 4 of the appendix.

4.1 RQ_1 - What tools for microservice reconstruction have been developed?

This question concerns a general description of discovered tools. We can summarize the following aspects:
Repository: Of 37 discovered tools, 2 are commercial and proprietary and do not share the code repository. Among the remaining 35 tools, 6 do not provide any kind of open repository.

Of the available tools, 2 were uploaded to Zenodo. This makes 'activity' metrics such as the number of Commits, Stars, and Forks impossible to infer and complicates potential development ('forking') by other researchers. Another 2 tools host codes on GitLab, with the remaining 25 on GitHub.
***License*:** The 2 commercial projects are covered by proprietary licenses, while 5 projects are not available at all.

The majority (17) of openly available projects do not indicate any license, which is a bad practice since it creates legal ambiguity about how the project can be used by third parties, which critically, in our case, includes other researchers.

The other 3 tools specify that they are available '`For Academic Use Only`,' which is better than not specifying any license, but also potentially ambiguous.

One project uses a `Creative Commons By-Attribution` license while CC licenses are not considered suitable for software even by the license authors [7].

The remaining are permissive OSS licenses - 7 instances of `MIT` License and 1 `Apache 2.0` License. Also, 1 instance of 'copyleft' `GPLv3` license is represented.
Activity: Most publicly available projects have not been updated since the tool/paper publication. Only 4 tools have commits in the first half of 2023, and additionally, 2 commercial tools are continuously supported. Another 8 tools were last updated in 2022. The earliest abandoned tool is Decomposer, which has not been updated since December 2016.

Another way to measure the activity of development is through commits. Most (16) tools for which we could gather such information have less than 100

[7] `https://creativecommons.org/faq/#can-i-apply-a-creative-commons-\ license-to-software`

Table 1: Summary of results

	RQ$_1$			**RQ**$_2$	**RQ**$_3$		**RQ**$_4$		
Tool name	Repo	License [1]	Last update	Platform [2]	Recovery [5]	Tool aims [6]	Input [3]	Output [4]	Visualization
Arcan	✓	P	-	Many[7]	S	SD	S	S, LV	✓
ARCHI4MOM	✓	-	06/22	OT	D	MR	T	OV	✓
Aroma	✓	M	04/23	OT	D	MR, SD	T	OV	✓
attack-graph-generator	✓	-	01/21	D	S	VD	D	HM	✓
Code2DFD	✓	AP	06/23	J	S	MR, VD	S	OV	✓
Decomposer	✓	-	12/16	J	S	M2M	MG	LV	-
IdentificationApproach	✓	-	01/22	J	S	M2M	S	SV	-
ImpactAnalysis	✓	-	01/19	J	S	Test	MG	T	✓
istio-log-parser	✓	-	05/22	I	D	MR	T	SV	✓
MAIG	-	-	-	OT	D	MR, P	T	SV, AP	✓
MicADO	✓	M	06/22	Any	H	MR	T, MC	SV	✓
microART	✓	A	04/17	D	H	MR	S,D,T	SV	-
MicroDepGraph	✓	-	11/21	J	S	MR	S	SV	✓
microFreshener	✓	M	11/22	Any	S	SD	MG	SV	✓
Microlyze	✓	-	07/18	EU, OT	D	MR	T	SV	✓
MicroMiner	✓	M	11/20	KU	H	MR	D	SV	✓
MAAT	-	-	-	J	D	MR, P	T	SV	✓
microserviceExtraction	✓	A	-	J	S	M2M	S	SV	✓
microTOM	✓	M	01/23	KU	H	MR	D	SV	✓
monitoring_ms	✓	-	08/18	EU,D	D	MR, MO	T	SV, HM	✓
mono2micro (socialsoftw.)	✓	M	06/23	J	H	M2M	S	SV	✓
Mono2Micro (IBM)	✓	P	-	J	D	M2M	T	SV	✓
MonoToMicro ([27])	-	-	-	J	S	M2M	S	SV	-
MS-MDE-RL	✓	G	01/22	J	S	M2M	MG	SV	✓
MSA-Nose	✓	-	04/21	J	S	SD	S	S	-
MSDesigner	-	-	-	J	S	M2M	MC	SV	✓
MSExtractor	-	-	-	J	S	M2M	S	SV	-
Rademacher et al. ([28])	✓	-	03/20	J	S	MR	S	DV, OV, SV	-
De Alwis et al. ([29])	✓	-	07/19	J	H	M2M	S,T	SV	-
Ntentos et al. ([30])	✓	CC	02/21	Any	S	MO	MC	Ref., Vio.	-
OpenTracingProcessor	✓	M	07/19	OT	D	MR, MO	T	SV, HM	✓
Prophet	✓	-	09/21	J	S	MR	S	SV	-
RAD	✓	-	01/21	J,P	S	MR, SEC	S	SV	-
ServiceCutter	✓	A	05/21	J	S	M2M	MG	SV	✓
Subtype	✓	-	05/18	Any	H	M2M	MG	SV	-
VECROSim	✓	-	12/22	KU	D	MO	MC	HM	-
VMAMV	✓	-	01/22	J	H	MO	S	HM	-

[1] P - Proprietary, M - MIT, AP - Apache v.2, A - Academic Use Only, G - GPL v.3, CC - CC BY 4.0

[2] OT - OpenTracing, D - Docker, J - Java, EU - Eureka, KU - Kubernetes, P - Python

[3] S - Source, T - Traces, D - Deployment files, MG - Model-generatable from source, MC - Model-custom manual format

[4] S - Smells, LV - Logical View, OV - Operational View, SV - Service View, DV - Domain View, HM - Health Metrics, T - Tests, AP - Anti-Patterns, Ref. - Refactorings, Vio. - Violations

[5] S - Static, D - Dynamic, H - Hybrid

[6] SD - Smell Detection, MR - Microservice Reconstruction, VD - Vulneraibility Detection, M2M - Monolith to Microservice, Test - Test Generation, P - [Anti-]Pattern Detection, MO - Monitoring, SEC - Security Analysis

[7] Java, C, C++, C#, Python

commits. We exclude from this 4 projects that have been pushed to a public repository without preserving the git history; thus, they had 1-2 commits.

The largest amount of commits are in `VMAMV` (798) and `mono2micro` (609); however, they are not that popular with the community as judged by Stars, Forks, as well as the number of citations among our papers in the online appendix [1].

The most popular by a huge margin, both in terms of GitHub stats and citations, is the `ServiceCutter`, which is one of the oldest projects in our list and is used as a reference implementation in many M2M papers. However, the project is now abandoned, and the build is broken, with only surviving Docker images making it possible to run it.

4.2 RQ_2 What languages/platforms are currently supported by tools?

The overwhelming majority of tools cover Java (21), most as the only supported language (19). Other represented languages include Python (2 tools), as well as C, C++, and C# (1 tool - Arcan is multiplatform). Apart from that, certain tools target a certain framework rather than a language - 6 tools use OpenTracing logs, 2 leverage Eureka, and 1 Istio. Some tools study deployments, with 3 tools studying Docker containers and another 3 Kubernetes pods. Additionally, 4 tools use some intermediate model representation as input, thus potentially being applicable to any language.

Figure 1 groups all available platforms hierarchically by corresponding input type (see Section 3.5/Table 1). It shows, for each specific platform (outer ring), how many tools support this platform. Note that, as explained above, some tools support several platforms, so numbers along the ring do not sum up to 37, and it is thus not possible to deduce intermediate categories from the figure and we do not put numbers to intermediate categories.

4.3 RQ_3 What is the purpose of reconstruction?

When it comes to Reconstruction approaches, 19 tools use Static methods and 10 Dynamic, with another 8 using a combination of both (Hybrid).

Some tools handle systems already using Microservice Architecture (17 in total). For some (10), reconstructing the architecture by providing the SDG is the only purpose. Another 2 tools couple this with Monitoring of the Microservice system, 2 with Pattern and 1 with Smell Detection, and 2 more with Vulnerability detection/Security Analysis; also 3 of found tools are concerned purely with Monitoring of the health of the system, and 3 more purely with Smell Detection.

Additionally, results included 12 tools that deal with Monolith to Microservice migration by analyzing the legacy monolithic system and proposing a grouping/refactoring of methods/classes into separate microservices.

Figure 2 shows the distribution of different purposes of reconstruction.

Fig. 1: Sunburst chart showing the correspondence between input types and platforms (languages). Some tools cover several languages, so numbers along the ring exceed 37.

4.4 RQ$_4$ What is the input/output of the tools?

The mapping between different types of inputs and outputs among the tools is presented in Figure 3.

As for the **input**, the majority (13) of the tools use source code as input directly, meaning they can be potentially integrated into IDEs or CI/CD pipelines. Further, 6 tools use some kind of generatable model that can be inferred from the Source, which can also be an automatic step in a pipeline (examples of models are `OpenAPI`, `microTOSCA`). Another 3 tools use a custom-defined Model format, which means that adoption is harder since tools to construct such models need to be developed first because manual model creation for large projects is impractical.

Additionally, 3 tools do not study the Source directly but instead use the Deployment files, which are usually checked into the same repository.

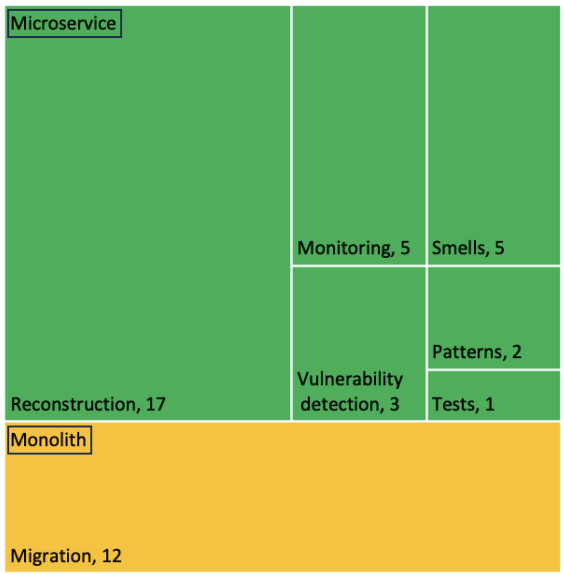

Fig. 2: Block diagram showing relative proportions of different reconstruction purposes. Note: some tools have several purposes, so numbers do not sum up to 37 tools.

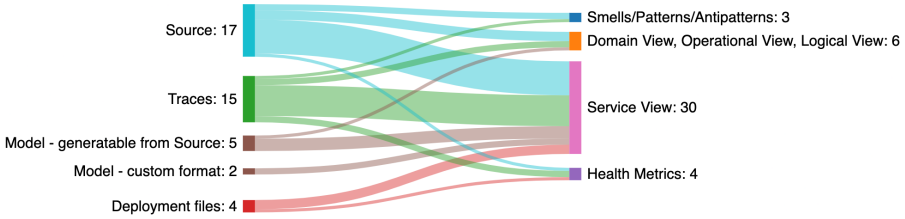

Fig. 3: Sankey diagram mapping tool Inputs to Outputs. Categories that appear only once are excluded; Views except Service View are grouped together.

Most of the tools that perform Dynamic Analysis use Traces (9), in particular, OpenTracing, with particular tools using either Jaeger[8] or Zipkin[9] as the tracing tool of choice, while others use Eureka[10] or Istio[11].

Furthermore, some tools combine different inputs (e.g., Traces and Sources).

[8] Jaeger `https://www.jaegertracing.io`
[9] Zipkin `https://zipkin.io`
[10] Spring Eureka Server `https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-eureka-server.html`
[11] Istio Service Mesh `https://istio.io`

When it comes to the **output**, the majority of tools return some kind of view (28), the most common being (as expected by the goal of this research) service view (26), which is either the Service Dependency Graph for Microservice systems (15) or Class to Microservice grouping for Monolith to Microservice migration systems (13[12]).

Additionally, 4 tools provide an operational view, 2 tools - a logical view, and 1 tool a domain view.

Out of 6 tools that deal with smell/pattern/anti-pattern detection (see previous RQ), 4 provide a list of detected smells/patterns/anti-patterns. However, the remaining 2 tools only report these detections on top of the provided SDG as part of the visualization, so we do not mark it as separately obtained input.

5 tools provide health metrics, which in particular can take the form of a system response to injected faults (2), an attack graph (1), or simply metrics given for different parts of the system (2). Another tool concerned with vulnerability detection returns a dataflow diagram and yet another a breakdown of roles required to access different endpoints to monitor potential privilege escalation problems.

Also, 2 tools aim to improve existing code - one by suggesting refactorings that solve detected violations, another by suggesting which CRUD-operations tests to implement.

Different tools use different output formats to provide the results - common include Neo4j, JSON, and microTOSCA to report SDGs or C2M mappings. Some tools only provide the reconstructed SDG as a graphic or web visualization in their front-end application. With 11 papers, we could not determine from the paper text or repository description what formats were used. Additionally, we could confirm that 23 tools provide some kind of visualization for their results while the remaining 15 either do not provide it or did not mention such support in documentation or source publication.

5 Discussion

Different architectural reconstruction tools, often with similar features, have been developed in the last few years. The analysis of the literature identified in the SMS indicates that while there is a need for MSA reconstruction tools, there is a limited amount of them actively developed, and in particular, there are no widely adopted tools.

Such tools are commonly built from scratch instead of extending previous ones. Some of the reasons why researchers are developing new tools might be the unavailability of tools meeting their requirements. Often, tools require particular resources, input, or configuration that discourage other research teams from using them. Another reason might be the impossibility of running them. Often,

[12] Note: one tool (MicADO) is reported with the aim being MR, but output being C2M because it studies an existing Microservice system and proposes a new, optimized grouping of methods into microservices

tools are not easily executable or require access to some libraries, databases, or specific hardware not available in the research group trying to run them. Moreover, not all the tools were available in source code repositories. To increase the availability of the tools, we recommend hosting the tools both on GitHub and archival platforms like Zenodo or Software Heritage, the latter providing easy integrations with the former.

It should also be emphasized that, when considering static analysis tools, only a limited number of tools are directly applicable to the industry, mainly because they parse a very limited number of languages or technologies. Most of the research-developed tools parse Java code, thus making them inapplicable in the industry where microservices are developed with a large number of technologies.

5.1 Future Research Directions

Based on the discussion of the research questions (RQs) above, we propose the following directions for future research in this field:

- Focus on validating existing tools and their outcomes to enhance their credibility and facilitate their adoption in the industry. While some tools already exist, our findings suggest that they have not undergone thorough validation in terms of precision and recall of the components of the SDGs, resulting in limited application.
- Center the tools around inputs that produce outcomes of genuine interest to stakeholders and explore the possibility of utilizing inputs from widely accepted technologies, both for static and dynamic analysis tools.

5.2 Threats to Validity

Various sources of bias or error could potentially impact the validity of our study's results. The research questions and classification schema used in our study may be subject to construction validity. To minimize this risk, the authors independently reviewed and discussed the research questions. As for the classification schema, we classified tools and their categories based on objective enumerated categories (e.g., language, license, etc.).

Also, to ensure replicability, we carefully identified and reported the bibliographic sources used to identify peer-reviewed literature. We also provided the search strings and the inclusion and exclusion criteria. Potential issues in the selection process could, however, arise from the choice of search terms, which may lead to an incomplete set of results. To mitigate this risk, we expanded the search string by including possible synonyms. Moreover, to address the limitations of search engines, we queried academic literature from four different bibliographic sources, and we performed both forward and backward snowballing [25] to increase the coverage of possible sources.

Other possible threats may apply to the reliability and generalizability of our results. As for reliability, all primary sources underwent review by at least

two authors to mitigate bias in data extraction, with any disagreements resolved through consensus involving a third author. As generalizability, instead, we mapped the academic literature on MSA reconstruction tools. However, we cannot claim to have screened all possible literature, as some documents may not have been appropriately indexed or may be subject to copyright restrictions or limited availability.

6 Conclusion

In this work, we performed a Systematic Mapping Study to classify the tools for MSA reconstruction. We classified 37 tools from 95 primary studies, comparing their input and output, which will be useful to researchers and practitioners to have a quick overview of the existing tools. It is interesting to note that the vast majority of tools are implemented from scratch without extending previous ones. Moreover, most tools are based on static analysis and can parse only a limited set of technologies.

We plan to extend this work by comparing the detection accuracy of the tools that can be executed on a set of microservice projects and conducting an industrial survey to investigate their applicability and the usefulness of the output provided.

Acknowledgements

This material is based upon work supported by grants from the Research Council of Finland (grants n. 349487 and 349488 - MuFAno) and 6GSoft project from Business Finland (grant n. 24304494 - 6GSoft); National Science Foundation (grant n. 2245287); and partly supported by the projects "FREEDA" (PRIN MUR, Italy, CUP: I53D23003550006) and "OSMWARE" project UNIPI_PRA 2022 64.

References

1. Bakhtin, A., et al.: Appendix to: Tools reconstructing microservice architecture: A systematic mapping study. Zenodo, `https://zenodo.org/doi/10.5281/zenodo.8207331`
2. Cerny, T., Abdelfattah, A.S., Maruf, A.A., Janes, A., Taibi, D.: Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study. Journal of Systems and Software **206** (2023) 111829
3. Baabad, A., Zulzalil, H.B., Hassan, S., Baharom, S.B.: Software architecture degradation in open source software: A systematic literature review. IEEE Access **8** (2020) 173681–173709
4. Medvidovic, N., Taylor, R.N.: Software architecture: foundations, theory, and practice. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2. (2010) 471–472

5. Perry, D.E., Wolf, A.L.: Foundations for the study of software architecture. ACM SIGSOFT Software engineering notes **17**(4) (1992) 40–52
6. Cerny, T., et al.: Microservice architecture reconstruction and visualization techniques: A review. In: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). (2022) 39–48
7. Abdelfattah, A.S., Cerny, T.: Roadmap to reasoning in microservice systems: A rapid review. Applied Sciences **13**(3) (2023) 1838
8. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: An update. Information and Software Technology **64** (2015) 1–18
9. Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). (2016) 44–51
10. Pahl, C., Jamshidi, P.: Microservices: A systematic mapping study. In: Proceedings of the 6th International Conference on Cloud Computing and Services Science - Volume 1 and 2. CLOSER 2016, Setubal, PRT, SCITEPRESS - Science and Technology Publications, Lda (2016) 137–146
11. Dragoni, N., et al. In: Microservices: Yesterday, Today, and Tomorrow. Springer, Cham (2017) 195–216
12. Soldani, J., et al.: The pains and gains of microservices: A systematic grey literature review. J. Syst. Softw. **146** (2018) 215–232
13. Neri, D., et al.: Design principles, architectural smells and refactorings for microservices: a multivocal review. SICS **35** (2020)
14. Ponce, F., et al.: Smells and refactorings for microservices security: A multivocal literature review. J. Syst. Softw. **192**(C) (2022)
15. Karabey Aksakalli, I., et al.: Deployment and communication patterns in microservice architectures: A systematic literature review. J. Syst. Softw. **180** (2021) 111014
16. Soldani, J., Brogi, A.: Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey. ACM Comput. Surv. **55**(3) (2022)
17. Bushong, V., Abdelfattah, A.S., Maruf, A.A., Das, D., Lehman, A., Jaroszewski, E., Coffey, M., Cerny, T., Frajtak, K., Tisnovsky, P., Bures, M.: On microservice analysis and architecture evolution: A systematic mapping study. Applied Sciences (Switzerland) **11**(17) (2021)
18. Gortney, M.E., et al.: Visualizing microservice architecture in the dynamic perspective: A systematic mapping study. IEEE Access **10** (2022) 119999–120012
19. Parker, G., et al.: Visualizing anti-patterns in microservices at runtime: A systematic mapping study. IEEE Access **11** (2023) 4434–4442
20. Abdelfattah, A.S., Cerny, T.: Roadmap to reasoning in microservice systems: A rapid review. Applied Sciences **13**(3) (2023)
21. Cerny, T., et al.: Microvision: Static analysis-based approach to visualizing microservices in augmented reality. In: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). (2022) 49–58
22. Walker, A., et al.: On automatic software architecture reconstruction of microservice applications. In: Information Science and Applications, Springer Singapore (2021) 223–234
23. Rademacher, F., et al.: A modeling method for systematic architecture reconstruction of microservice-based software systems. In: Enterprise, Business-Process and Information Systems Modeling, Springer International Publishing (2020) 311–326

24. Kleehaus, M., et al.: Microlyze: A framework for recovering the software architecture in microservice-based environments. In: Information Systems in the Big Data Era, Springer International Publishing (2018) 148–162
25. Wohlin, C.: Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In: International Conference on Evaluation and Assessment in Software Engineering. Ease '14 (2014) 1–10
26. Cerny, T., Abdelfattah, A.S., Bushong, V., Al Maruf, A., Taibi, D.: Microservice architecture reconstruction and visualization techniques: A review. In: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE), IEEE (2022) 39–48
27. Zaragoza, P., Seriai, A.D., Seriai, A., Bouziane, H.L., Shatnawi, A., Derras, M.: Refactoring monolithic object-oriented source code to materialize microservice-oriented architecture. (2021) 78 – 89
28. Rademacher, F., Sachweh, S., Zündorf, A.: A modeling method for systematic architecture reconstruction of microservice-based software systems. Lecture Notes in Business Information Processing **387 LNBIP** (2020) 311 – 326
29. De Alwis, A.A.C., Barros, A., Fidge, C., Polyvyanyy, A.: Availability and scalability optimized microservice discovery from enterprise systems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **11877 LNCS** (2019) 496 – 514
30. Ntentos, E., Zdun, U., Plakidas, K., Geiger, S.: Semi-automatic feedback for improving architecture conformance to microservice patterns and practices. (2021) 36 – 46