MiSAR: The MicroService Architecture Recovery Toolset

Nour Ali\textsuperscript{1}, Nuha Alshuqayran \textsuperscript{2}, Rana Fakeeh \textsuperscript{3}, Thoybur Rohman\textsuperscript{1} and Carlos Solis\textsuperscript{4}

\textsuperscript{1} Brunel University London, UK nour.ali@brunel.ac.uk, 2026156@brunel.ac.uk
\textsuperscript{2} Imam Mohammad Ibn Saud Islamic University, Saudi Arabia nshaqayran@imamu.edu.sa
\textsuperscript{3} AIDA Geschäftsführungs-Organisations-Systeme GmbH Hauptstr. 11, 75391, Germany rfakeeh@aidaorga.de
\textsuperscript{4} ION Group, UK carlos.solis@iongroup.com

Abstract. This tool demo paper presents the MicroService Architecture Recovery (MiSAR) toolset for software engineers (software architects and developers) that need to semi-automatically obtain as-implemented architectural models of existing microservice based systems. The MiSAR approach has been designed following Model Driven Architecture, and a set of components have been developed to support the semi-automatic support of MiSAR. The toolset first parses microservice based systems and generates a Platform Specific Model, which is an abstract representation of the system. Then, a model transformation engine automatically generates a Platform Independent Model which represents the as-implemented microservice architectural model. This model can also be introduced into the Graphical Model Generator where the software engineer can obtain quantitative metrics of the microservice architectural model and UML diagrams representing different views of the architecture.

Keywords: Microservice, architecture reconstruction, architecture recovery, architectural views, model driven engineering, model driven architecture.

1 Introduction

Microservice architecture has become a popular architectural style \cite{1}. Microservices are developed quickly and provide more agility to the system \cite{2}, which results in continuous architectural changes \cite{3}. Therefore, it can be stated that not every system is built using a well-documented architecture, and often the documentation of the architecture is not kept up to date \cite{4}. Keeping control of the overall architecture during development can be very difficult, especially when microservice based systems are designed, developed and deployed by different stakeholders and teams. Moreover, these architectures follow evolutionary design, which is very hard to manage, and architectural constraints are difficult to track. Software engineers often have little knowledge of the as-implemented architecture of their systems, and often face the challenge of not knowing in detail the underlying structures of the software system architecture.
The above concerns can be solved by using software architecture recovery (reconstruction or reverse architecting) [5, 6] which is a technique that reverse engineers systems to obtain the actual (as-implemented) architectural structure and description from system artefacts such as source code.

This paper presents the MicroService Architecture Recovery (MiSAR) toolset, which aims to support the architecture recovery of microservice systems by allowing software engineers to obtain semi-automatically an up-to-date architecture of implemented microservice systems. This can be challenging to obtain manually as microservices are not first-class citizens in the software, microservice systems use different programming frameworks and technologies, and microservices are highly inter-dependent, making analysis and architecture abstraction and comprehension difficult.

The MiSAR toolset, manuals, artefacts and its application to case studies can be found at [7]. The MiSAR toolset video demonstration is available at [8].

Fig. 1. MiSAR Model Driven Architecture abstraction levels

2 MiSAR Approach

MiSAR follows Model Driven Architecture (MDA) [9], to recover architectural models of existing microservice systems. The initial version of the MiSAR approach has been defined empirically in [10]. The MiSAR approach analyses the microservice software artefacts and produces models at two abstraction levels (see Fig. 1). Therefore, the approach includes the following MDA artefacts found at [7]:

- The Platform-Specific Metamodel (can be found at [7] as PSM.ecore file) defines the constructs which abstract microservice based systems using the platforms and technologies. For each microservice based system that needs to be recovered, a Platform Specific Model (PSM) is generated conforming to the Platform-Specific Metamodel. The current platforms and technologies which are supported are the Java Language, Docker and Spring boot framework and technologies which include Consul, Eureka, MongoDB, MySQL, Neo4j Graph database, OAuth2 and RabbitMQ.
• The Platform Independent Metamodel which defines the microservice architectural elements that describe a microservice architecture in a technology independent way. The metamodel includes 17 architectural element types which include Microservices that are classified into Functional Microservices, which realize the system’s business capabilities, and Infrastructure Microservices, which realize infrastructural capabilities. Infrastructure Pattern Components which support the functionality of patterns, Endpoints which are service URIs for remote calls and Service Dependencies which describe the communication between a consumer microservice and a provider microservice. Each architectural model recovered conforms to the Platform Independent Metamodel and is called a Platform Independent Model (PIM).

• Mapping Rules which map elements of PSMs into PIMs. Each mapping rule checks the existence of certain PSM elements, if they exist, then PSM elements are transformed into a group of target PIM elements. MiSAR currently supports 275 mapping rules.

One of the benefits for MiSAR in following the MDA approach is the separation of concerns. Models can be reusable and independent of their graphical notation. Also, an architectural model can be manipulated in other contexts and transformed into other forms.

3 Components of MiSAR Toolset

The MiSAR toolset is composed of three components which support a user to obtain an architectural model from a microservice system in a semi-automatic way. Each of the components, has as input and/or produces the MDA artefacts explained in section 2. The components in the toolset are the following:

• Parser: MiSAR includes a parser which statically analyses the source files of microservice based software. The parser currently analyses Java source code, configuration, Docker, Docker compose, build files (POM) at the microservice level and project build files at system level, which contain information used by Maven to build a project. The parser produces a Platform Specific Model (PSM) of the system by instantiating the Platform Specific Metamodel.

• Model Transformation Engine: MiSAR implements bottom-up model-driven transformations to obtain architectural models. PSMs generated by the parser, are fed into model transformations that automatically transform them into PIMs. The model transformations implement the mapping rules.

• Graphical Model Generator: To improve the understandability of the PIMs, we have developed a Graphical Generator. For each PIM, the generator creates: 1) metrics of the PIMs (architectural models) in excel sheets, e.g., a table with the number of architectural elements in an architectural model such as the number of microservices, pattern components and service dependencies, 2) images with graphical UML diagrams of the models and 3) PlantUML [13] files of the models. We currently use the UML Component diagram to represent the microservice architecture. The architecture can also have different views at architecture level and microservice level.
4 Implementation of MiSAR

The Platform Independent and Platform Specific Metamodels have been implemented as Ecore models using the Eclipse Modeling Framework (EMF) [11]. The MiSARParser is a python application that incorporates PyEcore, JavaLang, Yaml, XMLtoDict and other python libraries to parse YAML, XML and JAVA artefacts of a microservice-based application such as docker-compose.yml and pom.xml into a MiSAR PSM. The generated PSM is in Ecore (or XMI).

To automate the mapping rules, we have developed the model transformation engine using the Eclipse Model-to-Model Transformation (M2M) project, by incorporating the operational QVT transformation language (QVTo) [12]. The model transformation engine produces PIMs in Ecore (or XMI).

Finally, the graphical generator is a java application which navigates through PIMs and automatically translates them into UML graphical notations. The application uses the java Ecore implementations of MiSAR’s Platform Independent Metamodel and translates each element into PlantUML textual language [13] to create the images with the diagrams. The java application also creates excel sheets with metrics of the models.

5 A Walkthrough of MiSAR

We will demonstrate the steps and artefacts produced using the MiSAR toolset to semi-automatically generate the as-implemented architectural model of a microservice based system. To demonstrate MiSAR, we have selected a case study. The case study is an open-source project called the MicroCompany application [14]. MicroCompany is implemented using Java Spring Boot/Spring Cloud microservice-based application that consists of 11 microservices of which 4 are business oriented. It utilises both synchronous and asynchronous inter-service communication.

Consider that the software team, after having developed the MicroCompany application, would need to get an up-to-date architecture of their application. The software team have to follow the installation instructions and manual described in “ManualforM-iSARRecovery.pdf” found on [15] and follow the steps:

Step 1- Parsing the Microservice System to Create a PSM instance: The files from the MicroCompany GitHub are first downloaded locally. Then, the required artefacts are collected and uploaded to the existing MiSAR parser, as illustrated in Fig. 2, which shows the user has to input: the Project name, Path of the PSM Ecore metamodel file, Build directory of the system (multi-module) project, Path of every Docker Compose file (yml), Build directory of every microservice (single-module) project, Path of build file (POM) of the system (multimodule) project and the Path of the build file (POM) for every microservice (single-module) project. Configuration and Java Source artefacts are collected automatically by the parser with the help of the build directory of every microservice project.

The user has the option to delete or add files. This is to allow the user to control the parts of the system which they would like to recover. Users may want to recover the
architecture of the entire microservice system, whereas other users may want to only recover specific parts of the system, e.g., specific microservices.

![Fig. 2. User interface of Parser used to create PSM of MicroCompany](image)

The parser produces a PSM instance for the MicroCompany application. The PSM instance can be found at [16]. Even though the PSM is not the as-implemented architecture model, it is useful, as it provides backtracking support and allows the user to understand the elements that generated the PIM.

![Fig. 3. Example of the recovered “query-side-blog” micro service attributes](image)

**Step 2- Executing Model Transformations to Create the PIM instance:** The PIM architectural model is obtained by running the Eclipse QVTo project. The PIM recovered is in XMI format and can be opened as a tree view with Sample Reflective Ecore Model editor provided by the Eclipse Modeling Framework (EMF).

In addition, the microservice view has the attributes for the microservices. Fig. 3 shows attributes for a recovered microservice called query-side-blog. For instance, (a) the “query-side-blog” microservice exposes an endpoint with request URI “GET /blogposts/search/findByDraftTrue” which is handled by (b) the service operation “findByDraftTrue()” and (c) returns a response service message of model “Page(BlogPost)”.

![Fig. 3. Example of the recovered “query-side-blog” micro service attributes](image)
As it can be noticed, one of the attributes is “Generating PSM” which indicates the element from the PSM that was used to generate the attribute. This feature provides traceability and backtracking support for the recovery.

![Graphical Model Generator](image)

**Fig. 4. Using the Graphical Model Generator for MicroCompany**

**Step 3- Transforming the PIM XMI into Graphical Architectural Diagrams:** Once you have a PIM instance, the Graphical Model Generator can be used. The user selects the PIM instance and indicates the location where the different images and excel sheets will be located once produced (see Fig. 4). Then, automatically a drag down menu with all the microservices of the architectural model of the PIM instance will be visible under Microservice Level. The user can produce images with UML architecture diagrams and metrics at the architecture level or at a microservice level as follows:

- **At Architecture Level:** If the user clicks on the Architecture Metrics Excel Data sheet, an excel sheet is produced that contains the number of architectural elements for every single architectural element type. Fig. 5 shows the excel sheet produced for MicroCompany. For example, there are 5 Functional Microservices and 6 Infrastructure Microservices in MicroCompany. In addition, the user can click under Dependency View and create an image (Download PNG and Download SVG buttons) or get the PlantUML file for the graphical UML diagram. Fig. 6 shows the dependency diagram for the architecture of MicroCompany. The diagram shows the microservices of the architecture and their dependencies. Blue components are Functional microservices and purple components are Infrastructure microservices.

![Architecture Level Metrics](image)

**Fig. 5. Architecture Level Metrics of recovered MicroCompany**
At microservice Level: As it can be noticed from Fig. 6, it is very hard to read the architectural diagram of a medium to large architectural model such as MicroCompany. Therefore, the tool allows the user to select from the top-down menu a specific microservice. Once a microservice is selected, they can create an excel with metrics for that microservice, a microservice view which shows the pattern components, endpoints and service interfaces and a microservice dependency view diagram which shows the microservice chosen and the service dependencies it has with others. Fig. 7 shows that Circuit-Breaker microservice has 4 Pattern Components, 1 Infrastructure Service Component, 2 Infrastructure Client components, 1 Service Interface, 1 endpoint and 7 Service Dependencies. Fig. 8 shows the microservice dependency view diagram for Circuit-Breaker. Circuit Breaker has 7 dependencies with other microservices.

6 Evaluation

In this section, we evaluate the components of the tool to demonstrate the time it takes for MiSAR to generate the as-implemented architecture (the PIM instance) for three open-source projects. Table 1 shows the time it takes, for each toolset component, on an Intel Processor Core(TM) i5-7200U CPU @ 2.50GHz, 2701 Mhz, 2 Core(s), 4 Logical Processor(s). The time for the Graphical Generator Component is not shown as this is instantaneous. It can be noticed that for a large project, such as TrainTicket, the parser
takes most of the time of the recovery process. However, several days could have been taken if software engineers would want to recover the architecture manually. Manual recovery could also produce an inaccurate architecture due to human errors or an architecture with not enough details.

Table 1. Time of MiSAR toolset to obtain as-implemented architecture models.

<table>
<thead>
<tr>
<th></th>
<th>LOC</th>
<th>Parser to generate PSM (sec)</th>
<th>Model Engine to Transform PSM to PIM (sec)</th>
<th>Total No of Recovered elements in PIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>MicroCompany</td>
<td>127.1K</td>
<td>9</td>
<td>3.89</td>
<td>490 including 11 microservices</td>
</tr>
<tr>
<td>TrainTicket [17]</td>
<td>507.2K</td>
<td>446</td>
<td>63.15</td>
<td>1341 including 69 microservices</td>
</tr>
<tr>
<td>MusicStore [18]</td>
<td>116.6K</td>
<td>1</td>
<td>1.07</td>
<td>107 including 9 microservices</td>
</tr>
</tbody>
</table>

7 Related Work

One of the few existing works related to ours is MicroART [19]. MicroART also uses model-driven engineering but does not follow MDA. In MiSAR, the architectural model is recovered automatically from the PSM, i.e., there is no human input, whereas in MicroART, a software architect needs to identify service discovery services. MiSAR produces architectural models that are richer than MicroART as MicroART only has 8 architectural concepts whereas MiSAR has 17.

MicroLyze [20] is another work which proposes an architecture recovery approach for microservices. MicroLyze, unlike MiSAR, does not adopt a model-driven approach. Instead, it utilises a distributed tracing component that dynamically monitors simulated user requests. In addition, the work of Wang et al. [21], present an automated recovery process using system source code to build a dependency graph. Like MiSAR, their approach is based on source code analysis. However, their approach does not employ model-driven architecture and does not recover many elements such as patterns.

8 Conclusion

In this paper, we have introduced the MiSAR toolset that semi-automatically generates as-implemented architectural models from existing microservice systems implemented in diverse technologies. We have demonstrated how the MiSAR toolset components can be used and its evaluation on 3 microservice projects. Our further work includes improving the usability aspects of the toolset and the efficiency of the parser. In addition, we are working on extending MiSAR to support its analysis of additional languages and technologies. We also plan to evaluate our approach with practitioners.
References

8. MiSAR Toolset video demo: https://youtu.be/sdRdKLesyS0