MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-based Systems

Giona Granchelli*, Mario Cardarelli*, Paolo Di Francesco†,
Ivano Malavolta‡, Ludovico Iovino†, Amleto Di Salle*
*University of L'Aquila, L'Aquila, Italy
{giona.granchelli | mario.cardarelli}@student.univaq.it, amleto.disalle@univaq.it
†Gran Sasso Science Institute, L'Aquila, Italy - {paolo.difrancesco | ludovico.iovino}@gssi.it
‡Vrije Universiteit Amsterdam, The Netherlands - i.malavolta@vu.nl

Abstract—**Microservice-based systems are characterised by a multitude of small services, each running in its own process and communicating with lightweight mechanisms. The microservice architectural style strongly encourages high decoupling among microservices in order to ease their independent deployment, operation, and maintenance. However, there are situations in which having a global overview of the system is fundamental. In this paper we present the first prototype of our Architecture Recovery Tool for microservice-based systems called MicroART. MicroART following Model-Driven Engineering principles, is able to generate models of the software architecture of a microservice-based system, that can be managed by software architects for multiple purposes.**

Index Terms—**Microservices, Architecture Recovery, MDE.**

I. Introduction

Many companies are adopting microservice architectures (MSAs) for their applications on the basis of all the benefits they can provide, e.g. technology heterogeneity, resilience, scaling, ease of deployment, composability [7]. Despite some general architecture recovery approaches already exist [8], in the microservices area little investigation is being performed in the field. This is confirmed by a recently published mapping study on architecting microservices [3], which has reported limited research focus on the activities related to MSA recovery. Since microservices can dynamically change their status at runtime (e.g., IP address) for multiple reasons (e.g., autoscaling, failures, upgrades), it is common in these architectures to adopt service discovery mechanisms in order to keep the system dependencies loosely coupled. Service discovery services are used to allow each microservice in the network to dynamically discover the other active services. Although service discovery mechanisms simplify many aspects of microservice communications, from the architecture recovery point of view they create additional challenges as they mask the real dependencies existing among microservices. This creates several difficulties when trying to understand the details and the dependencies of the system.

In this paper we present MicroART, a microservice architecture recovery tool[1] capable of recovering the system physical and logical architectures and generate model-based representations. MicroART is capable of generating to different architectural model, respectively referred as *physical* and *logical* architecture models. The *physical architecture model* is the abstraction of the system recovered by the architecture recovery activity. The architectural model in which service discovery services are resolved is the *logical architecture model*. Moreover by *service discovery resolution* we mean the process of identifying the real targeted microservice of a communication as if the service discovery was not acting as a proxy of the communication. Both the physical and the logical architectural model are snapshots in time of the system.

Of the main features of MicroART is its ability to identify and remove service discovery services, with the main aim of unveiling the real dependencies between microservices from the development perspective. MicroART has been realized using Model-Driven Engineering tools and development principles [2]. We have designed a minimal domain specific language (DSL), based on EMF[2], that can be used to describe microservice architectures. Moreover, the physical and logical models extractions are respectively obtained by using *text-to-model* and a *model-to-model* transformations. The service discovery resolution can be seen as model refactoring, where the models can be managed by using a graphical editor, built on the top of EuGENia[3].

The rest of the paper is organized as follows. Section II presents the overview of the MicroART approach. Section III discusses the details of the MicroART tool. Section IV describes the uses and potential impact of the tool. Section V reports the related work. Section VI discuss the conclusions and future work.

II. The MicroART approach

An overview of the activities, tools, and artefacts involved in the activities performed by MicroART is depicted in Figure 1. MicroART recovers the architecture of a microservice-based system by applying three different phases.

Physical Architecture Recovery. This phase is responsible to perform all the operations needed to recover the *model* of the physical architecture of the system.

Service Discovery Identification. In this phase the software architect can access and reason on the generated model of the physical architecture, where identify and mark the *service*

[1]Available for download at: https://github.com/microart/microART-Tool

[2]https://eclipse.org/modeling/emf

[3]www.eclipse.org/epsilon/doc/eugenia

discovery services, and then update the model with this information. By resolving service discovery services, MicroART is able to understand dependencies among services. The interaction with the architect is required in order to guarantee the correctness of the service discovery identification.

Logical Architecture Recovery. In this phase MicroART takes as input the updated model and uses the information provided by the software architect to resolve service discovery services and generate the *logical model* of the architecture. The software architect can use this model for several purposes, for instance: documentation, architectural analysis, or transform it into another formalism.

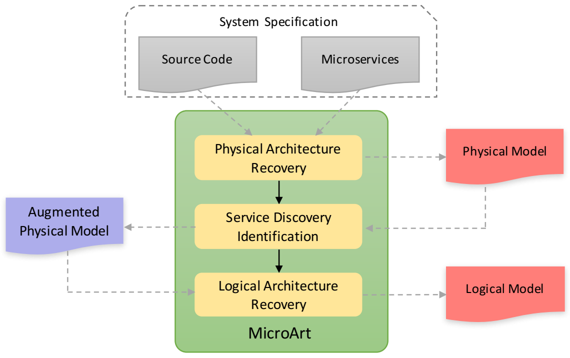

Fig. 1. Architecture recovery activities, tools and artefacts

The graphical interface of MicroART has been realized by using Epsilon and EuGENia, a model-driven tool that automatically generates an interactive modeling editor for manipulating the models in a user-friendly manner.

III. Realizing the MicroART tool

MicroART is composed of four main components, each one addressing a separate task, as highlighted in Figure 2.

The first component is the *GitHub Analyzer*, which takes as input the web URL of the system's source code repository, it clones locally the repository and extracts the information related to: *(i)* the system name and description, and *(ii)* the developers which have contributed to the repository over the software lifetime. The second component is the *Docker Analyzer* that dynamically queries the Docker runtime environment and gathers the network interface and the IP address of each services. This information is used to map the services information gathered from the *GitHub Analyzer* with the respective network identification, otherwise we cannot access to this specific information.

The *Log Analyzer* is responsible for analyzing the log files that are generated dynamically by a monitoring tool. At the moment, MicroART relies on the log generated by TcpDump[4], an open-source tool for capturing network traffic and for packets analysis. TcpDump is used to register and write log files of the communication requests collected during the system execution. Using these logs, the Log Analyzer can track the communications among services.

The *Model Log Analyzer* takes as input the model of the physical architecture upon which the service discovery services have been identified, and uses this information to properly filter the log files. From this analysis, the model of the logical architecture of the system is generated.

A first assessment of the MicroART tool has been performed using an open-source benchmark system called AcmeAir[5], a microservice-based implementation of a fictitious airline website. AcmeAir uses Docker container technology, and it contains six microservices: *Main*, *Authentication*, *Customer*, *Booking*, *Flights*, *Nginx*, which are all developed in JavaScript and running in a Node.js runtime environment. In AcmeAir, microservices are not aware of each other in the network, but they communicate with the service *Nginx*, which acts as a service discovery service. In the following sections we present each phase of the MicroART tool in detail and we discuss how it has been applied on the AcmeAir benchmark system.

A. Physical Architecture Recovery

In order to start the recovering of the physical architecture, MicroART needs the following inputs: *(i)* the GitHub repository URL of the system, *(ii)* the log files of the communication among microservices, and *(iii)* Docker runtime environment information (i.e., containers identifiers, the IP addresses and the network interface used by the service).

The GitHub Analyzer uses the url in order to clone the system's repository and parse the content. The GitHub Analyzer searches for the following information: *(i)* a *Docker-compose file* specifying the system components interactions (e.g., container name and build-path), and *(ii)* a *Docker-file* for each microservice from which specific listening ports and exposed ports are retrieved. In addition the GitHub Analyzer retrieves the list of all the developers that have contributed with at least a commit operation to the repository. The information discussed above is retrieved statically, i.e., without running the system, but it is not enough for generating the physical architecture of the system. The information collected with the *GitHub Analyzer* is sent to the *Docker Analyzer* which will query the Docker environment at runtime in order to retrieve

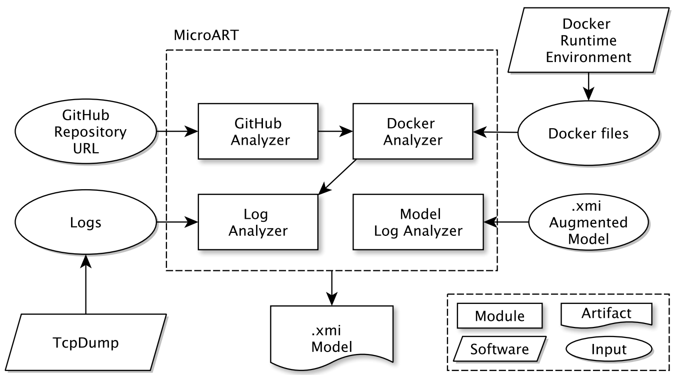

Fig. 2. MicroART inner structure

[4]http://www.tcpdump.org/
[5]https://github.com/acmeair/acmeair-nodejs

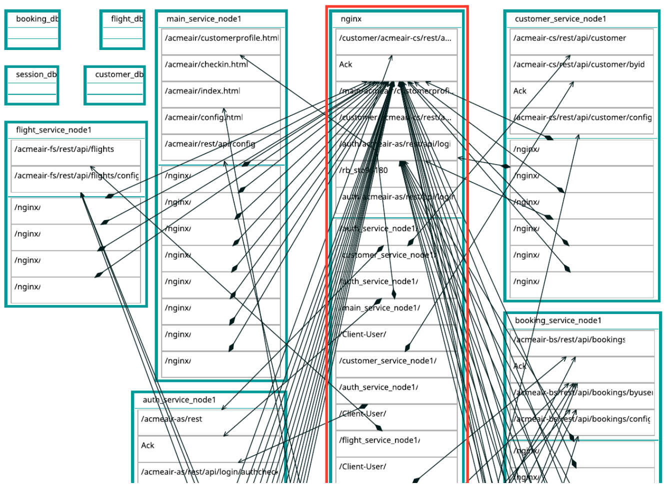

Fig. 3. AcmeAir physical architecture model generated by MicroART (extract)

information related to the IP address of every microservice and the network interface used by each microservice. This additional data is fundamental to accomplish the last task, because MicroART needs to know how the microservices interact with each other. One of the ways to retrieve this information is to monitor the system dynamically in its running environment and to save microservice communications into log files. By joining the information gathered from the *static analysis* (i.e. repository analysis) and the *dynamic analysis* (i.e. runtime analysis) MicroART can generate the model of the physical architecture. Figure 3 shows the physical architecture model of the AcmeAir benchmark system recovered by MicroART. Ten elements have been extracted: six microservices and four databases. Each service has its own set of interfaces which is used for connecting to the others services. Each interface represents either an incoming or an outgoing communication and each communication link represents a *resource dependency*.

B. Service Discovery Identification

In the second phase towards recovering the logical architecture of a microservice-based system, a software architect manual interaction is required. By using the developed graphical editor, the software architect is able to visualize the model generated by MicroART and identify and mark service discovery services. Figure 3 reports a screenshot of the physical architecture rendered in our Eclipse-based editor. The service discovery service identified by the software architect has been highlighted in red. As expected, the service discovery service is receiving and forwarding all the communications to and from most microservices.

C. Logical Architecture Recovery

MicroART performs a new log analysis on the *augmented physical model* for extracting the logical architecture. At this point MicroART knows all the communications among microservices and the identified service discovery service. The procedure performed in this phase is similar to the physical recovery phase, except for the fact that the communications in which the service discovery service are involved are analyzed differently. Every link to the service discovery service is traced to the real target of the communication. At this point, the physical model is transformed in the logical model by: (i) removing all the previous links among microservices and replacing them with the newly identified connections, and (ii) removing all service discovery services identified in the physical model. In order to connect the service consumer to the real service provider, MicroART checks the log of each registered communication. Each communication starts with a client requesting a resource and routing this resource to the service discovery, which in turn forwards the request to the microservice providing the resource. In this phase, MicroART knows which one is the service discovery and can realize a mapping of microservices and communications traces, from the beginning until the end, and for each communication, a link is created to the connected microservices.

Figure 4 reports a screenshot of the AcmeAir logical architecture model, where the service discovery has been removed and the number of dependencies have significantly been reduced. Indeed, *Main* does not need any resource from the other services, except for the rendering of its web page. Similarly, *Flight* relies only on its own database, and does not need to communicate with other components. From Figure 4, we can notice that some microservices (i.e., *Authentication*, *Customer* and *Booking*) are dependent from each other. The software architect can use the resulting logical architecture for many purposes, like change impact analysis at the architectural level, deep understanding of the overall architecture of the system, etc.

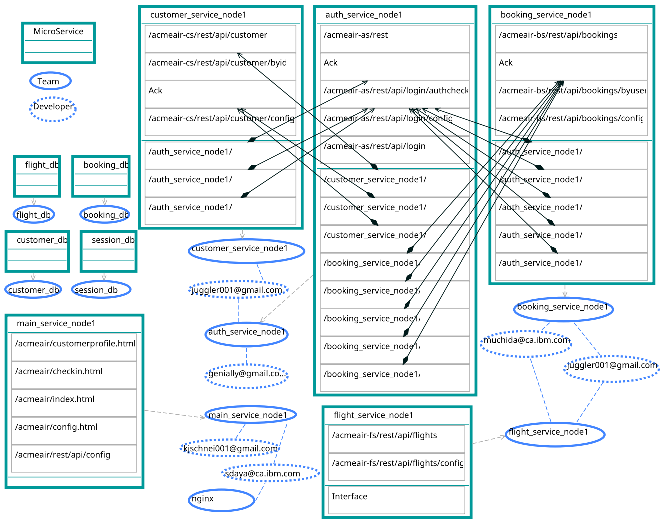

Fig. 4. AcmeAir logical architecture model generated by MicroART

If we compare the AcmeAir physical (Fig. 3) and logical (Fig. 4) architecture models, we can immediately notice how the latter is easier to understand and analyze. Indeed, while in the physical model the microservices are all attached to the service discovery, in the logical architecture the microservices are connected directly to each other. The reason why service discovery services must be identified and removed is because they mask the flow of operation for the communications.

IV. Uses and Potential Impact

The intended users of MicroART are public or private companies that are interested in automatically recovering the physical and logical architectures of their microservice-based systems. MicroART is a *semi-automatic* tool for architecture recovery where the only manual software architect interaction is the identification of the service discovery services.

At the moment, the MicroART prototype depends on GitHub and Docker, and its applicability is restricted to projects based on these technologies. The main benefit in the adoption of MicroART is the implementation of a model-based representation of the microservices architecture. The models generated by MicroART can be graphically rendered, and thus be used for many purposes, such as documentation, architectural analysis, architectural reasoning or verification between the deployed architecture and the designed one. Since this tool has been built around microservices systems needs and it considers the famous approach *you build, you run*, also the developed DSL considers the concepts of *teams* and *developers*, being the microservice approach organized with cross-functional teams around services [5]. Indeed the tool graphically represents developers and teams, allowing software architects to better understand the team responsibilities and the mapping of microservices with the respective developers. Additionally, due to the fact that microservice architectures are usually composed by numerous services, and the interactions among them are frequent, the number of connections is generally significant. A graphic rendering of the architecture could reveal important information, as for instance the *decentralized data management* and *decentralized governance* [5], which may lead to the proliferation of a high number of microservices within a single system.

V. Related Work

A SOA-oriented architecture recovery process called QAR (QUE-es Architecture Recovery) is presented in [1]. Similarly to MicroART is based on both a static and dynamic phases but it is based on a set of tools, relying on UML to understand the system details. X-Trace presented in [4], is a tracing framework for reconstructing a comprehensive view of services behaviour. This tool monitors a distributed system and tries to extract the element behaviours starting from a network interface. Despite X-Trace monitors the network, X-Trace doesn't rely on models but it relies on metadata, and doesn't produce a logical architecture. In [6] a comparative analysis of software architecture recovery techniques is presented. They argue that most accurate technique are ARC and ACDC. ARC relies on a program's semantics to perform recovery and represents a software system as a set of documents. Differently, ACDC works on software written in C language, but is not applicable to microservice architectures, because is a clustering technique for architecture recovery that relies on monolith system.

VI. Conclusions and Future Works

In this paper we presented MicroART, a prototypical tool for architecture recovery of microservice-based systems. MicroART first generates a physical architectural model of the system, then it refines it to a logical architecture model. Both generated models can be used by software architects for many purposes, for example architectural reasoning, analysis, or documentation. In the future, we are planning to extend MicroART to support other container-based engines (e.g., Vagrant), to support the integration with additional logging and rendering tools, and to investigate the application of MicroART to other microservice technologies (e.g., AWS Lambda serverless Function-as-a-Service). We will apply MicroART on additional open-source microservice-based systems in order to refine and improve its capabilities.

VII. Acknowledgment

This research has been supported by the European Union's H2020 Programme under grant agreement number 644178 (project CHOReVOLUTION - Automated Synthesis of Dynamic and Secured Choreographies for the Future Internet), and by the Ministry of Economy and Finance, Cipe resolution n. 135/2012 (project INCIPICT - INnovating CIty Planning through Information and Communication Technologies).

References

[1] F. Cuadrado, B. García, J. C. Dueñas, and H. A. Parada. A case study on software evolution towards service-oriented architecture. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1399–1404. IEEE, 2008.

[2] A. R. da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems and Structures*, 43:139 – 155, 2015.

[3] P. Di Francesco, P. Lago, and I. Malavolta. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. *IEEE International Conference on Software Architecture (ICSA)*, to appear, 2017.

[4] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, pages 20–20, 2007.

[5] M. Fowler and J. Lewis. Microservices a definition of this new architectural term. *URL: http://martinfowler.com/articles/microservices.html*, 2014.

[6] J. Garcia, I. Ivkovic, and N. Medvidovic. A comparative analysis of software architecture recovery techniques. In *Automated Software Engineering (ASE), IEEE/ACM 28th International Conference on*, pages 486–496. IEEE, 2013.

[7] S. Newman. *Building Microservices*. O'Reilly Media, Inc., 2015.

[8] L. O'Brien, C. Stoermer, and C. Verhoef. Software architecture reconstruction: Practice needs and current approaches. Technical report, DTIC Document, 2002.

Appendix

MicroART is a prototypical **research tool** available for download at: https://github.com/microart/microART-Tool. Even if the tool is still in a prototypal stage and continuously evolving, we managed to test and validate the approach on a system called AcmeAir, an publicly available open-source benchmarking system. We will extend the evaluation to other systems soon, hopefully in collaboration with industrial partners.
The demonstration will be carried out using two projectors. On one side, labeled with Ⓐ, we will show every step required to recover the architecture from a GitHub repository, on the other side (labeled with Ⓑ) we will show the models generated by MicroART in a step-by-step fashion.

In the following we provide the main steps of the tool demo:

1. *Introduction to MicroART*
   A short introduction about the tool will be provided in order to show its main components and the architecture recovery process.

2. *Configuration setup*
   The *Acme Air* benchmark system will be run on the top of the Docker platform environment in order to give an idea about its main services and features.

3. *Running MicroART*
   MicroART is composed of two Eclipse instances. One instance deals with the steps of architecture recovery, the other instance hosts the plugins responsible for graphically rendering the models of the recovered architecture. The architecture recovery will start by providing as input the GitHub URL of the Acme Air repository. MicroART will automatically clone the repository and generate an initial version of the architecture of the system.

4. *Monitoring and simulation*
   A monitoring tool for logging the requests among microservices, called TCPDump, will be launched.
   A brief usage scenario of the Acme Air system will be performed in order to generate requests among microservices. In the meanwhile, TCPDump will store those requests into specific log files.

5. *Physical Architecture Recovery*
   MicroART will generate the architecture model in the format presented in section III and the model will be rendered in Ⓑ.

6. *Service Discovery Identification*
   Upon the model rendered in Ⓑ, the service discovery service will be identified, and the model will be updated accordingly.

7. *Logical Architecture Recovery*
   Using the architecture model generated in step 5 and the information provided on the service discovery in step 6, MicroART will generate the logical model of the architecture. The model will be rendered in Ⓑ.

8. *Future work*
   Currently, MicroART is a prototype implementation and we are planning to extend MicroART to support other container-based engines alternative to Docker(e.g., Vagrant), and the integration with additional logging and rendering tools. Further application of MicroART will be performed on other microservice-based systems in order to refine and improve its capabilities.

9. *References*
   Finally, the references to MicroART will be shown to allow the audience to know where they can download the tool and also find more details about its implementation and benchmarks.

Depending on the time availability, some of the aforementioned steps may be shortened or deleted.