SAVAT: A Tool for Visualizing the Impact of Changes in Microservices*

Adam Kattan Rahmani[1], Gabriel Elijah Goulis[2], David Kozák[3,4], Tomas Cerny[2], and Tomáš Vojnar[1,3]

[1] FI, Masaryk University, CZ [2] SIE, University of Arizona, US
[3] FIT, Brno University of Technology, CZ [4] Oracle Labs, CZ

Abstract. Change impact analysis is a vital process for managing changes in complex systems. This becomes especially important in decentralized systems like those based on microservices. The paper presents a SAVAT tool designed to facilitate change impact analysis in microservice systems. SAVAT enables practitioners to interact with visual exploration of microservice systems, both from two architectural views (service and domain views), using the established notions of context maps and service dependency graphs, along with a novel notion of inter-microservice call graphs. Our tool can also automatically highlight changes in different versions of microservice systems and provide support to explore how the changes propagate through the system.

1 Introduction

Microservices have gained substantial popularity in the industry due to their advantages in scalability, flexibility, and reliability. Decomposing monolithic applications into independently evolvable units offers significant flexibility to developers. However, this architectural shift makes it more difficult to maintain a comprehensive, holistic view of the system [8]. Even minor changes in one part of the system can trigger ripple effects, potentially leading to major failures [5].

A similar kind of issue appears also in object-oriented programs where the *change impact analysis* (CIA) approach has been proposed to provide feedback on the semantic impact of program modifications [15]. More recently, CIA techniques specifically tailored for microservice systems have emerged [6]. However, to the best of our knowledge, currently there is no general-purpose tool that offers an intuitive interface to holistically explore the system structure.

This paper introduces SAVAT (Software Architecture Visualization and Analysis Tool), a tool for visualizing the impact of changes in microservice systems[1]. SAVAT enables practitioners to interactively explore both the *service* and the *domain* architectural views, represented by the established notions of *service dependency graphs* (SDG) and *context maps* (CM), respectively. In addition, SAVAT introduces *inter-microservice call graphs* (IMCGs), an extended form of classical call graphs that captures both local method invocations within a single microservice and remote calls between methods in different microservices. This allows for a more detailed exploration of the dependency

* The work was supported by the Czech Science Foundation project 23-06506S, the FIT BUT project FIT-S-23-8151, and the FI MU project MUNI/A/1600/2024. This material is also based upon work supported by the National Science Foundation under Grant No. 2409933.

[1] The SAVAT showcase video can be found at https://youtu.be/yR_YqUXRKjs.

chains compared to the SDG alone. Furthermore, SAVAT offers automated CIA capabilities in all its views, enabling users to compare two versions of a system, identify differences, and analyze how local changes will affect the overall system.

The visualization offered by SAVAT provides stakeholders with multiple views on both the overall architecture of the system and the results of the analyses carried out on it. This allows stakeholders to identify architectural antipatterns, such as low cohesion, high coupling, or cyclic dependencies. Addressing these issues, stakeholders can propose architectural changes that eliminate potential antipatterns and reduce the risk of bugs or the reappearance of antipatterns in future versions of the microservice system.

2 Preliminaries

Fundamental concepts and notation include microservices, software architecture reconstructions, and change impact analysis.

2.1 Microservices Architecture

Systems designed using *microservice architecture* (MSA) consist of multiple autonomous services. Each of these services, known as a microservice, communicates with others via various types of API [7]. A single microservice can be described as a distinct deployment unit. Ideally, microservices should have a single responsibility and be contained within a specific context [13]. Their size may also be influenced by the technologies used, the types of API communication, and the deployment strategies [14].

2.2 Software Architecture Reconstruction

Software architecture reconstruction (SAR) is an approach that can be used to extract key knowledge about a given microservice system, which can later be used to analyze the impact of changes in the system [11]. In particular, we use this process to generate multiple views on the given microservice system, including its context map (CM), service dependency graph (SDG), and inter-microservice call graph (IMCG).

For our purposes, we use a tool called Prophet [3], which employs GraalVM Native Image [17], an ahead-of-time optimizing compiler. This tool is capable of extracting the necessary knowledge from each microservice for the construction of the views mentioned above. The extracted knowledge includes information on REST calls, endpoints, entities, and call graphs (CGs) that characterize the given microservice system. Prophet consists of two parts: the first part is a plugin within GraalVM Native Image that performs the extraction, while the second is a dedicated Java application that runs GraalVM, along with the plugin, on top of each microservice to create multiple views from the previously extracted knowledge.

The results of SAR can be presented from various perspectives. These perspectives may include views centered on entities and bounded contexts (CMs), views that focus on communication between different microservices (SDGs, IMCGs), and views that examine the internal implementation of a specific microservice (microservice CGs).

To illustrate each type of SAR result, we begin with *SDGs*. These graphs demonstrate how microservices communicate with each other through API calls. The edges of the graph represent every pair of REST calls and endpoints between the microservices. These pairs are formed by matching the target URI of the REST call with the URI of the

endpoints, as well as by matching the HTTP method of the REST call with the HTTP method of the endpoint. We focus on REST calls at the moment as they are provided by Prophet out of the box, but SAVAT is extensible in this regard.

CMs provide a perspective that emphasizes the relationships between entities within a microservice system. An entity is defined as an object that contains data and revolves around a specific use case. Each entity has a unique identifier that sets it apart from other entities in the CM. These entities can persist over time, and their structure may be based on a template, like a class in Java.

A *CG* is a directed graph whose nodes represent the methods and the edges indicate their caller-callee relationships [9]. Visualizing CGs provides users with an efficient way to navigate the intricate relationships between calls. Below, we distinguish between two types of CGs: *individual microservice CGs* and the *IMCG* that combines individual microservice CGs with the SDG. This graph not only illustrates the caller-callee relationships among all methods in the microservice system but also highlights the API calls between different microservices, linking them to their specific methods.

2.3 Change Impact Analysis

Minor changes in projects written in object-oriented languages may have major and global effects. Similar problems arise in microservice systems, made even worse by their distributed nature. Indeed, a change in one microservice may break another distant microservice and cause a ripple effect [6]. The goal of the CIA is to provide feedback on the semantic impact of a set of program changes [15], enabling stakeholders to address unwanted changes caused by the ripple effect.

The first step in CIA is to identify changes within a defined artifact. According to [15], in object-oriented programming, changes can be categorized into various types of atomic changes. These atomic changes can be classified into three main categories: changes made to classes (such as adding or removing methods or fields), modifications to the body of methods, or the behavior of method lookups, and changes that involve the creation or deletion of a class. This categorization can naturally be extended to microservice systems too.

3 Capabilities of SAVAT

We now describe the key capabilities of SAVAT that allow practitioners to assess the impact of changes in microservice systems. The figures presented are based on data obtained from running SAVAT on the well-established Train Ticket benchmark [18].

3.1 Change Impact Analysis in SAVAT

In SAVAT, CIA is performed only on CMs and IMCGs. We do not perform CIA on SDGs since they are subsumed by IMCGs. Performing CIA in SAVAT involves multiple steps. At first, we collect the changed artifacts. A changed artifact is either a method whose bytecode hash differs or an entity with a modified field or field annotation.

After assessing the changes, the CIA begins to examine each change and trace its propagation. This tracing relies on connections defined by the model being used. For example, when tracking changes made to methods, the process follows the caller-callee relationships within the call graph. In this case, it makes sense to perform a type of reverse graph traversal starting from the initially changed method and collect all the methods that are traversed, as each of them may eventually call the initially changed method and therefore might be influenced by its change.

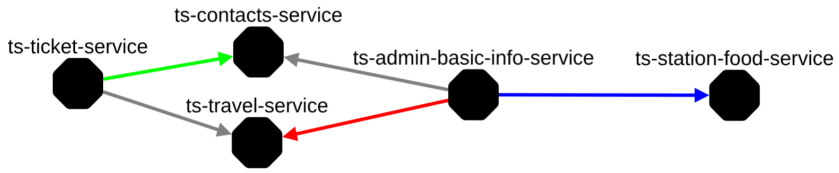

Fig. 1: An example of visualization of the LDA conducted on two different SDGs.

3.2 Links Difference Analysis in SAVAT

Unlike CIA, *links difference analysis* (LDA) operates exclusively with edges. The visualization highlights edges that have been added (shown in green), modified (shown in blue), and deleted (shown in red). This visualization is available only for CMs and SDGs. In SAVAT, users can easily access the LDA visualization using a utility bar located above the visualized graph.

Figure 1 shows an example of what the visualized result of LDA can look like:

- The green edge indicates that a new connection has been added between *ts-ticket-service* and *ts-contacts-service*, which means that at least one REST call was added to *ts-ticket-service* that invokes an endpoint of *ts-contacts-service*.
- The red edge shows that the connection between *ts-adminbasic-info-service* and *ts-travel-service* has been completely removed, meaning that all REST calls from *ts-admin-basic-info-service* that invoke endpoints from *ts-travel-service* were deleted.
- The blue edge indicates a modified connection between *ts-adminbasic-info-service* and *ts-station-food-service*, showing that at least one REST call has been changed between them.

3.3 Antipatterns and Metrics

SAVAT is capable of not only demonstrating the impact of changes introduced in a microservice system, but also of identifying and visualizing various types of *antipatterns* and *metrics*. The antipatterns that SAVAT can visualize include *high coupling* and *cyclic dependencies*. The high coupling antipattern identifies which microservices have a number of connections exceeding a user-defined threshold. The cyclic dependencies antipattern highlights the microservices that are part of a cycle in the visualized graph.

3.4 Dependency Analysis in SAVAT

A rendered SDG can provide key insight into the dependency network drawn from the architecture and how service dependencies evolve over time. The graph structure can be traversed manually. During that, selecting nodes will highlight direct dependencies and provide context on how the services are connected. Since changes to these dependencies may induce breaking changes, our visualization brings attention to modifications made specifically to the rest calls that form the dependencies alongside endpoints targeted by rest calls. In addition to highlighting direct connections, SAVAT provides the ability to visualize the communication graph up to the depth provided. This makes the impact of modifications on downstream dependencies and on the overall system more obvious.

Furthermore, SAVAT provides automated *reachability analysis*, as depicted in Figure 2. When a user clicks on a given node, the node itself and all transitively reachable nodes are highlighted in yellow.

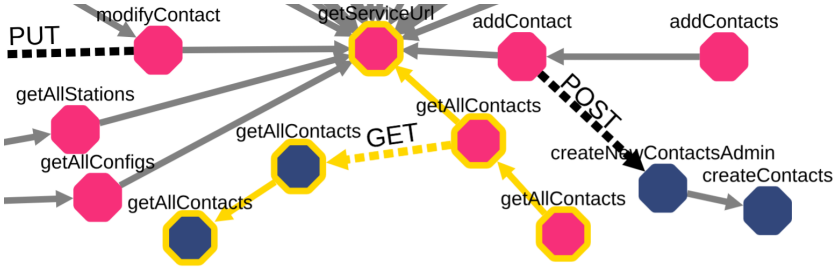

Fig. 2: Highlighted reachability of *getAllContacts* methods.

3.5 Inter-Microservice Call Graph

Visualizing connections between microservices typically offers a broad and holistic perspective. However, to gain deeper insights, it is often more effective to visualize the interactions between individual methods. The real challenge lies in accurately representing the *caller-callee relationships* between the methods located in different microservices. To address this, we propose a more detailed view, called the inter-microservice call graph (IMCG).

In this directed graph, nodes encode individual methods, distinguished by color according to their microservice. The edges denote caller-callee relationships that are either within a single microservice or across microservices via API calls, capturing the precise connections between methods in a way reflecting real inter-service communication.

SAVAT also supports CIA on IMCGs. The resulting graph visually displays nodes with green, blue, or red dotted borders, indicating the type of change that happened: in particular, green is used for addition, blue for modification, and red for deletion. The graph also includes other nodes and edges that depend on the changed node. Figure 3 shows the result of a CIA conducted on two versions of IMCG where all three types of changes are visible:

- *Modification* – indicated by the blue dotted border around the *queryForTravel*, *queryById*, and *queryByIdBatch* methods. The modification of *queryForTravel* affected another method of the same name that serves as an endpoint handler. This, in turn, affected methods in different microservices that invoke the endpoint via a REST call using the POST HTTP method. For example, among the impacted methods are the *preserve* methods.
- *Addition* – indicated by the green dotted border around the *createEvent* methods. The *queryByIdBatch* method, which was modified due to calling the newly added *createEvent* method, in turn impacted its direct caller, the *queryForNameBatch* method.
- *Deletion* – indicated by the red dotted border around the *updatePriceConfig* method. The deletion impacted not only methods within the same microservice, but also methods in other microservices.

This CIA result clearly shows that adding, deleting, or modifying a method can impact many other methods, not only within the same microservice but also across multiple microservices. Another observation is that the modified method *queryById* is not called by any other method.

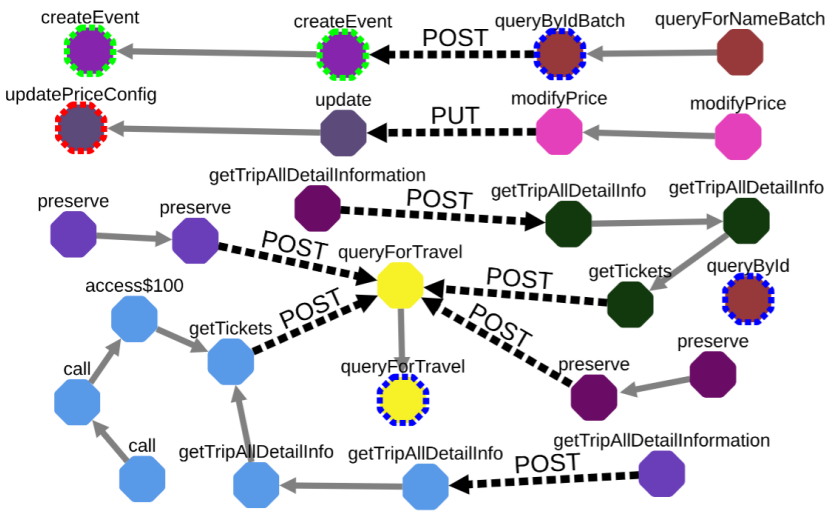

Fig. 3: A visualization of the CIA conducted on two different versions of IMCGs.

4 Architecture and Implementation of SAVAT

This section details the architecture of SAVAT in two parts: the first part describes the extraction of preliminary data for visualization, and the second part explains how the visualization process subsequently processes and displays this data.

4.1 Intermediate Representation

Before starting with visualization, it is necessary to extract relevant data from the analyzed projects and store them in a form that allows efficient automated analysis. SAVAT uses a hierarchical JSON-based intermediate representation (IR) originally introduced in [1]. It is extracted from the source code and describes holistically the structure of the microservice system. It is the foundation for visualizing how the system evolves by tracking how the source code changes over time. It can be broken up into code components, which are source code structures, e.g. method declarations or method calls, and an additional layer of context, which provides information on the microservices present along with metadata on the overall system like the commit id.

The IR is created by iterating over project files and mapping source code to code components. Contextual information like the individual microservices is based on package management files such as Gradle or Maven files. This is done over a series of commits, forming a temporal data set that tracks changes in architecture and implementation over time. The extraction process leverages the JavaParser library that forms AST structures from source code based on the Java language syntax (https://javaparser.org). The JGit library (https://github.com/eclipse-jgit/jgit) is used to interface with Git version control within Java, to locally clone projects, and to iterate over their version history.

4.2 Visualization

The related tools for visualizing the SAR data, which we present below, use libraries that render the data in a 3D space. In contrast, SAVAT uses a different library called

Cytoscape.js, which focuses on graph visualization in a 2D space [16]. This library provides a variety of layout algorithms, of which SAVAT specifically uses the fCoSE layout algorithm [2]. This algorithm effectively calculates the positions of the nodes and provides various attributes that allow further customization of the layout structure.

Each input or analysis output graph is managed through its own Cytoscape instance, which contains information about: (1) Graph elements objects (nodes and edges) with their metadata. (2) The container component in which the graph is visualized. (3) Styles that define colors, shapes, and labels for graph elements. (4) Layout options, including the layout algorithm and its attributes. These instances facilitate effective and customizable visualizations of graphs. They are then used in React.js (https://react.dev/) components with embedded data and a selected layout algorithm.

SAVAT utilizes several modern tools to create an intuitive user interface (UI). This user interface facilitates graph data visualization and offers users robust data management options, including the ability to add new graph data sources, remove existing ones, filter the data, and perform analyses.

To implement the described functionality, SAVAT uses several helper tools. Axios (https://axios-http.com/docs/intro) is a promise-based HTTP client that allows SAVAT to make calls to the back-end REST API. Shadcn (https://ui.shadcn.com/docs) is a library of UI components that provides various ready-to-use components. Tanstack Query (https://tanstack.com/query/latest/docs/framework/react/overview) is a library that facilitates asynchronous fetching, caching, and the creation of state or mutation hooks for making REST API calls to the back-end.

5 Related Works

Microvision [4] is a static analysis tool that brings visualization and observability to microservice architectures by using augmented reality. It leverages augmented reality to make viewing densely packed systems easier to manage. This seeks to address a well-known concern in visualizing microservice systems: as the system expands, it degrades visibility and makes navigating the system hard. Microvision functions by extracting a model of parsed source code and then, through SAR, in multiple steps of extraction, parsing, and manipulation. The visualization can be manipulated and traversed in 3D space and provides pop-up displays of relevant information while navigating the graph.

Harris et al. [10] designed a prototype tool that compares 2D and 3D visualization techniques with static analysis. SAR is used to generate an intermediate model from source code that is visualized by the tool. The visualization features intractable nodes that pop up relevant information and search features to navigate the system. A survey of users revealed that 3D scenes may be hard to traverse due to scale and could unnecessarily prolong observation of the system.

Another tool capable of SAR is described in [12], which employs profiling tactics in order to extract knowledge about a given microservice system.

6 Conclusion and Future Work

Although microservice systems provide significant advantages and have gained traction, managing architectural degradation in these systems has become challenging. With SAVAT, we bridge the gap between what developers do and the consequences of their

changes. SAVAT is capable of providing extensive impact analysis in distributed MSA's by visualizing structural changes to the system.

In future versions of SAVAT, we aim to provide customization of the user interface and plan to incorporate more data into the existing visualization.

Data Availability. SAVAT is open source and is available in the following git repository https://github.com/katyadam/SAVAT. Moreover, all the examples presented are based on the openly available train ticket benchmark, as mentioned above.

References

1. Adams, L., S. Abdelfattah, A., Hossain Chy, M.S., Perry, S., Harris, P., Cerny, T., Amoroso d'Aragona, D., Taibi, D.: Evolution and Anti-patterns Visualized: MicroProspect in Microservice Architecture. In: Proc. of ECSA'23. LNCS, vol. 14590. Springer (2023)
2. Balci, H., Dogrusoz, U.: fCoSE: A Fast Compound Graph Layout Algorithm with Constraint Support. Transactions on Visualization and Computer Graphics 28 (2021)
3. Cerny, T., Abdelfattah, A.S., Bushong, V., Al Maruf, A., Taibi, D.: Microservice Architecture Reconstruction and Visualization Techniques: A Review. In: Proc. of SOSE'22. IEEE (2022)
4. Cerny, T., Abdelfattah, A.S., Bushong, V., Al Maruf, A., Taibi, D.: Microvision: Static Analysis-based Approach to Visualizing Microservices in Augmented Reality. In: Proc. of SOSE'22. IEEE (2022)
5. Cerny, T., Chy, M., Abdelfattah, A., Soldani, J., Bogner, J., et al.: On Maintainability and Microservice Dependencies: How Do Changes Propagate? In: CLOSER'24. Scitepress (2024)
6. Cerny, T., Goulis, G., Abdelfattah, A.S.: Towards Change Impact Analysis in Microservices-based System Evolution (2025), https://arxiv.org/abs/2501.11778
7. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: Yesterday, Today, and Tomorrow. Springer (2017)
8. Granchelli, G., Cardarelli, M., Francesco, P., Malavolta, I., Iovino, L., Di Salle, A.: MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems. In: Proc. of ICSAW'17. IEEE (2017)
9. Grove, D., DeFouw, G., Dean, J., Chambers, C.: Call Graph Construction in Object-Oriented Languages. In: Proc. of OOPSLA'97. ACM (1997)
10. Harris, P., Gortney, M., Abdelfattah, A.S., Cerny, T., Rivas, P.: Designing a System-Centered View to Microservices Using Service Dependency Graphs: Elaborating on 2D and 3D visualization. In: Proc. of ICECCME'24. IEEE (2024)
11. Hutcheson, R., Blanchard, A., Lambaria, N., Hale, J., Kozak, D., et al.: Software Architecture Reconstruction for Microservice Systems Using Static Analysis via GraalVM Native Image. In: Proc. of SANER'24. IEEE (2024)
12. Nakazawa, R., Ueda, T., Enoki, M., Horii, H.: Visualization Tool for Designing Microservices with the Monolith-First Approach. In: Proc. of VISSOFT'18. IEEE (2018)
13. Newman, S.: Building Microservices: Designing Fine-Grained Systems. O'Reilly (2021)
14. Pahl, C., Jamshidi, P.: Microservices: A Systematic Mapping Study. In: Proc. of CLOSER'16. ACM (2016)
15. Ryder, B.G., Tip, F.: Change Impact Analysis for Object-Oriented Programs. In: Proc. of PASTE'01. ACM (2001)
16. Shannon, P., Markiel, A., Ozier, O., Baliga, N.S., Wang, J.T., Ramage, D., Amin, N., Schwikowski, B., Ideker, T.: Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. Genome Research 13(11) (2003)
17. Wimmer, C., Stancu, C., Hofer, P., Jovanovic, V., Wögerer, P., Kessler, P.B., Pliss, O., Würthinger, T.: Initialize Once, Start Fast: Application Initialization at Build Time. In: Proc. of OOPSLA'19. ACM (2019)
18. Zhou, X., Peng, X., Xie, T., Sun, J., Xu, C., Ji, C., Zhao, W.: Benchmarking Microservice Systems for Software Engineering Research. In: Proc. of ICSE'18. ACM (2018)