MicroKarta: Visualising Microservice Architectures

Oscar Manglaras
University of Adelaide
Adelaide, Australia
oscar.manglaras@adelaide.edu.au

Alex Farkas
University of Adelaide
Adelaide, Australia
alex.m.farkas@gmail.com

Peter Fule
Swordfish Computing
Adelaide, Australia
peter.fule@swordfish.com.au

Christoph Treude
Singapore Management University
Singapore, Singapore
ctreude@smu.edu.sg

Markus Wagner
Monash University
Clayton, Australia
markus.wagner@monash.edu

ABSTRACT

Conceptualising and debugging a microservice architecture can be a challenge for developers due to the complex topology of inter-service communication, which may only apparent when viewing the architecture as a whole. In this paper, we present MicroKarta, a dashboard containing three types of network diagram that visualise complex microservice architectures, and that are designed to address problems faced by developers of these architectures. Initial feedback from industry developers has been positive. This dashboard can be used by developers to explore and debug microservice architectures, and can be used to compare the effectiveness of different types of network visualisation for assisting with various development tasks.

CCS CONCEPTS

• **Human-centered computing** → **Interface design prototyping**; *Information visualization*; • **Software and its engineering** → Publish-subscribe / event-based architectures.

KEYWORDS

Microservices, Microservice Architectures, Network Visualization

ACM Reference Format:
Oscar Manglaras, Alex Farkas, Peter Fule, Christoph Treude, and Markus Wagner. 2024. MicroKarta: Visualising Microservice Architectures. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering (FSE Companion '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3663529.3663808

1 INTRODUCTION

Microservice architectures are a form of software architecture that has seen a growth in popularity over the past decade. Ideologically, microservices follow the paradigm of "do one thing and do it well", making each service small and simple [4]. However, in doing so, much of the complexity of the architecture instead shifts to how those microservice communicate and collaborate with each other [1]. Developers can hence benefit from visualising these high level connections.

In this paper we present MicroKarta, a software tool that visualises microservice communication topologies using three complementary network diagrams; a node–link diagram, an adjacency matrix, and an arc diagram, as well as a number of associated interactive features. MicroKarta is designed to be used as reference documentation, as well as to help with exploratory analysis and debugging. The most distinguishing feature of MicroKarta is the focus on supporting channel information in publish/subscribe communication architectures, which Aksakalli et al. [2] found was the second most common microservice communication paradigm. This is in contrast to the more common RESTful communication paradigm, which Microkarta also supports.

MicroKarta builds on our previous study [10] where we asked 20 developers of an industry microservice-based project to rate the impact and frequency of various development problems. The most impactful and frequent of these problems (Table 1) were the focus of the visualisation and interactive features developed for the tool. MicroKarta is not intended to address all the problems identified in [10], but we are taking the study into account for all of our decisions.

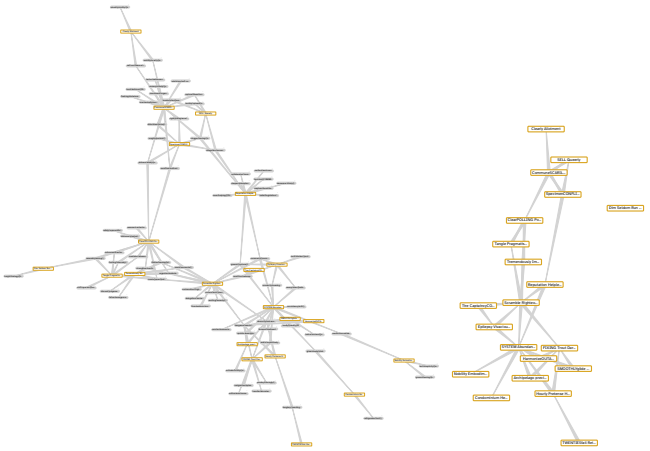
The most frequent problems were largely related to the topology of connections between microservices and channels, problems for which network diagrams are especially well suited. We hypothesise that MicroKarta's different visualisations will each be better suited

Table 1: Top six problems ranked by impact and frequency [10].

	Impact		Frequency
I1	cause of faults	F1	data structures of channels
I2	data structures of channels	F2	channels connected to microservices
I3	services affected by faults	F3	microservices sending data to a microservice
I4	services affected by message broker faults	F4	microservices receiving data from a microservice
I5	data structures used by microservices	F5	microservices connected to a channel
I6	purpose of the microservice	F6	microservices data structures

FSE Companion '24, July 15–19, 2024, Porto de Galinhas, Brazil

ACM ISBN 979-8-4007-0658-5/24/07
https://doi.org/10.1145/3663529.3663808



(a) With both microservice and channel nodes visible.

(b) With channel nodes hidden.

Figure 1: Node–Link diagram. The gold nodes are microservices and the grey are channels.

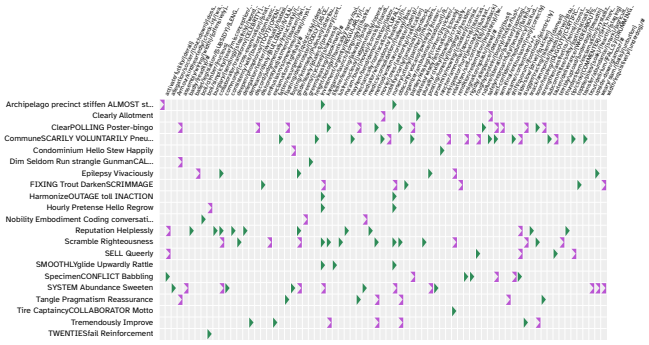


Figure 2: Adjacency Matrix. The rows are microservices and the columns are channels. The green wedges represent publish relationships, and the purple sockets represent subscribe relationships.

to addressing different problems and tasks. This tool could ideally be used in future research to investigate that question.

This work is being co-funded by Swordfish Computing, and as such MicroKarta is closed source and under a proprietary license. Swordfish's 100 microservices are split and re-used between multiple architectures and were created by around 25 developers. Swordfish has a team of around 25 developers who support approx. 100 microservices. These microservices are reused between multiple architectures, which primarily use publish/subscribe communication and contain, on average, 20 running services. All of the networks visualised in this paper are real (obfuscated) microservice architectures in use at Swordfish.

2 VISUALISATION DESIGN

We chose network diagrams as the basis for our visualisations because most of the highest frequency problems identified by [10] were related to identifying the connections between microservices (F2, F3, F4, and F5), which can be presented with network diagrams.

2.1 Node-Link Diagram

[Addresses F2, F3, F4, F5]

The node–link diagram is shown in Figure 1. There are two views, one showing channels, and one where the channel nodes are hidden. To provide information about the channels being published and subscribed to (F2 & F5), we are using two different types of node. The gold nodes are the microservices, and the gray nodes are the channels. The channel nodes were made the same colour as the links in an attempt to make the microservice nodes stand out. The view with the channels hidden is equivalent to the node–link diagram of a synchronous or RESTful communication architecture, and may be better suited to addressing problems F3 & F4.

For the links we used tapered lines, with the broad end representing the source of the link, and the narrow end representing the target. This follows the recommendation [8] that tapered edges provide better comprehensive performance than conventional arrows. We employed this design after experiencing difficulties reading large networks in early prototypes. At the time of writing, the SVG spec does not supported lines with varying widths (though there is a proposal[1]). However, the effect can be replicated with a simple isosceles triangle that has a base perpendicular to the slope of the line between the nodes.

The implementation uses force-based algorithm to automatically place the nodes. The user can then use the mouse to drag and pin nodes in place on the canvas to improve upon the default layout.

2.2 Adjacency Matrix

[Addresses F2, F3, F4, F5]

The adjacency matrix visualises the microservices in a tabular view as shown in Figure 2. We assigned each microservice a row and each column a channel. We can visualise all communications this way because microservices are only communicating with each other through channels; if we needed to visualise non-channel communication as well, such as REST requests, then we add extra columns to represent the synchronous microservices. There is another view that abstracts out the channels.

To differentiate between publish and subscribe behaviour, we used different coloured shapes in each cell. A green wedge represents a publish relationship, and a purple socket represents a subscribe relationship. The shapes were chosen to provide a contextual hint about the type of relationship they represent, with the wedge inserting (publishing) data into the socket. The colours were chosen to add to the visual distinctiveness of the two shapes, and to aid in preattentive searching [14, Ch. 5].

It has been confirmed by our industry partner that services can both publish and subscribe on the same channel. We need to use different symbols for publish and subscribe that can fit in the same cell to handle this case.

The matrix presents all the same information as the node-link diagram, and hence addresses the same problems, but it is more

[1] https://www.w3.org/Graphics/SVG/WG/wiki/Proposals/Variable_width_stroke

Figure 3: The arc diagram. Full names available on hover.

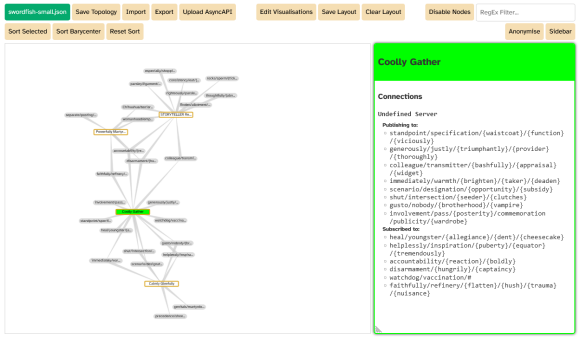


Figure 4: Sidebar showing info for a selected microservice.

compact and can be sorted. The downside is that it is harder to follow links and identify clusters in the data [12].

2.3 Arc Diagram

[Addresses F3, F4]

The arc diagram (Figure 3) is a node-link diagram arranged across a single dimension. It strikes a balance between the node-link diagram and the adjacency matrix; it shows direct links between nodes, but has an inherent, sortable order, like the matrix. The direction of the links is denoted with tapered lines, as with the node-link diagram, as well as denoted by the placement of the arc above or below the horizontal axis.

Unlike the previous two visualisations there is no channel information, meaning it only addresses problems F3 and F4. However, we hypothesise that the arc diagram is superior at presenting a directional path through the topology due to the ability to lay the nodes out in a one–dimensional ordered list.

2.4 Sidebar

[Addresses I5, I6, F1, F2, F3, F4, F5, F6]

The sidebar (Figure 4) provides a way to present detailed information about a specific microservice. The only information we are currently displaying in the sidebar is the name of the service and the channels being published and subscribed to (F2). However, it could also be used to address impactful problems that are difficult to address with network diagrams, for example I1 and I5 (data format schemas); I1, I3, and I4 (service fault details), and I6 (a description of the purpose of the microservice).

2.5 Reachability Highlighting

[Addresses I1, I3]

Reachability highlighting displays the data flow paths to and from a selected node (Figure 5). The highlighted paths represent the possible data flows through the topology. It can be used to find

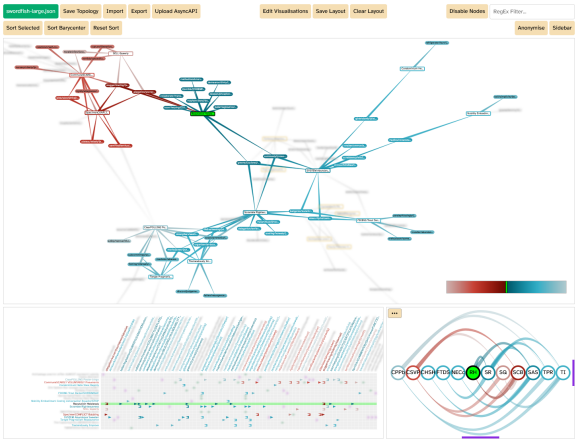


Figure 5: Reachability highlighting for the selected node (green). The blue nodes are 'downstream', reachable from the selected node. The red nodes are 'upstream', the selected node can be reached from them. Nodes that are not reachable are blurred and faded (and hidden in the arc diagram).

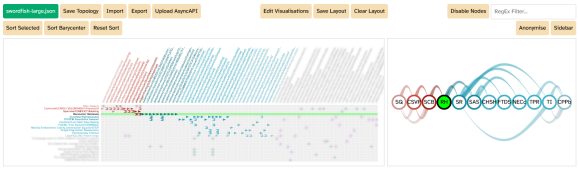


Figure 6: Sorting the matrix and arc diagram based on reachability to and from the selected node (compare to Figure 5).

the possible upstream cause of a microservice fault and identify which other down stream services might be affected by faults in the selected microservice

2.5.1 Reachability Sorting. To make the reachability highlighting more useful when applied to the matrix and the arc diagram, we have implemented a sorting option based on the distance of nodes from the selected node. Figure 6 shows the same architecture and selected microservice as Figure 5, but with the sorting applied. The arc diagram seems especially well suited to visualising reachability as a flow of data through the selected node, and for detecting cycles in the graph (the inverted arcs in Figure 6 indicate potential cycles). This could be something to investigate in a future study.

3 IMPLEMENTATION DETAILS

MicroKarta is implemented as a web app. All the visualisations and interactive features are run locally in the browser. A (NodeJS[2]) backend is used purely to store architectures between sessions. The architectures are represented by JSON objects that can be stored as files.

The visualisations are all developed using the D3[3] JavaScript library. D3 provides a powerful, low-level, API for creating and

[2] https://nodejs.org/
[3] https://d3js.org/

manipulating SVG graphics [3]. D3 was one of the main reasons we chose to use web technologies for this dashboard, the other reason being the platform-agnostic nature of web apps.

The layout of the three visualisations uses the CSS grid layout[4]. The visualisations are not directly placed onto the grid; instead we specify the number of vertical and horizontal cells each visualisation spans using the `span` keyword with the `grid-row-end` and `grid-column-end` properties. The browser automatically places the visualisations without overlaps and automatically updates placements if the span values change, which we can do on mouse events with some simple JavaScript. This is a simple way to automatically place visualisations in a way that looks (in our opinion) decent, produces predictable output, and allows dynamic resizing.

4 INDUSTRY FEEDBACK

To get feedback on our visualisations, we distributed the dashboard to three microservice developers at Swordfish. They use AsyncAPI[5] to define the publish/subscribe information for their microservices, so we wrote an algorithm that takes uploaded AsyncAPI files and outputs the architectural information in the JSON format the dashboard expects. Similar algorithms could be written to support other filetypes, but that is beyond the scope of this paper. The developers were asked to use the dashboard to explore their architectures and given a set of feedback questions to answer afterwards[6].

On the whole, developers found the dashboard useful, stating that: "the dashboard/tooling fulfils its purpose in my view, visualising complex AsyncAPI projects", and: "the diagram is useful for understanding complex systems, I see it being used as a way to discover bugs before they are pushed into production". Another clarified that, "our [architectures] are now getting to a size where its not possibly to hold in your head all the flows between the services... I can also see it being useful for investigation issues and tracking some bugs, and also in design phases where it's difficult to get a feel for the complexity".

When asked which of the visualisations they found most useful, participants were split between the node-link diagram and the adjacency matrix. One found the node-link diagram the most useful and the matrix the second, stating that, "the force diagram was helpful to get an overall view of the system and the matrix was useful for getting a more in-depth view of each service and topic". The other two found the matrix more useful; one paired the matrix with the sidebar stating that, "it was the easiest way to see all of the detail I wanted to see in a single view", while the third stated that they "found the matrix diagram useful, the force and arc not so much". However, they went on to clarify that that was "down more to [their] use case at the moment, which is identifying if there are any oddities in the async apis", they thought the force and arc diagrams would be more useful for "trying to diagnose issues".

All of the participants found the arc diagram to be the least useful. However, the developers were using the tool primarily to explore their existing architectures, they were not using it to help with debugging tasks, which the arc diagram is intended to address.

However, the feedback was not all positive, the developers criticised the difficulty and "friction" in finding and collecting all of the AsyncAPI files from the various microservice repositories, with one developer suggesting we, "look to supplement that workflow with automated collection of these files".

5 LIMITATIONS AND THREATS TO VALIDITY

At present, our feedback comes from only three developers at a single company; a wider range of feedback would be more representative of the industry as a whole. Additionally, the developers only had access to the tool for a short period of time; extended use may uncover further strengths and weaknesses in the tool design. The closed-source nature of the tool makes it difficult for others to test our designs.

6 RELATED WORK

We have not been able to find any published works that attempted to to visualise the channels in a publish/subscribe communication paradigm. However, looking more broadly, there are a number of works that have visualised other communication paradigms.

MicroART [7] is a tool for recovering the architecture of microservice systems and visualising connections between microservice endpoints. However, the visualisations are static. Engel et al. [5] and Mayer & Weinreich [11] both present interactive dashboards to visualise microservice architectures. Both use node-link diagrams to visualise the connections between services. Mayer & Weinreich's dashboard attempts to provide a wider range of information through multiple views and visualisations; it covers a wider range of information than our dashboard, including runtime data. MICROLYZE [9] uses an adjacency matrix to show microservice dependencies and instances. Microvision [15] is an attempt to adapt a node-link topology visualisation to augmented reality. Frisell [6] designed an interactive dashboard with custom visualisations, similar to a node-link diagram, but where the microservices are clustered based on the developer maintaining the service. Microusity [13] is a tool that tests RESTful APIs and generates graphs of service invocations, which received positive user feedback.

7 CONCLUSION

Developers of microservice architectures can benefit from network visualisations of the connections between services. Existing academic work in this area have each presented just a single type of network diagram, either node–link diagrams or adjacency matrices, and do not attempt to visualise channel information for publish/subscribe architectures. MicroKarta provides three types of network diagram with a focus on visualising channels and is designed to address the results of our previous study into microservice development problems. Initial industry feedback was positive, highlighting its use as an exploratory tool, but suggested the tool would be improved by automating the collection of the data needed to produce the visualisations, which currently needs to be input manually. The tool opens the door to future studies to determine which of the three types of network diagram are best suited for addressing different tasks.

Demo: https://universityofadelaide.box.com/v/microkarta-demo

[4] https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout
[5] https://www.asyncapi.com/
[6] https://doi.org/10.5281/zenodo.11044837

REFERENCES

[1] Carlos M. Aderaldo, Nabor C. Mendonça, Claus Pahl, and Pooyan Jamshidi. 2017. Benchmark Requirements for Microservices Architecture Research. In *2017 IEEE/ACM 1st International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*. 8–13. https://doi.org/10.1109/ECASE.2017.4

[2] Işıl Karabey Aksakalli, Turgay Çelik, Ahmet Burak Can, and Bedir Tekinerdoğan. 2021. Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software* 180, 111014 (Oct. 2021). https://doi.org/10.1016/j.jss.2021.111014

[3] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2301–2309. https://doi.org/10.1109/TVCG.2011.185

[4] Thomas F. Düllmann. 2017. *Performance anomaly detection in microservice architectures under continuous change.* Master's thesis. University of Stuttgart. https://doi.org/10.18419/opus-9066

[5] Thomas Engel, Melanie Langermeier, Bernhard Bauer, and Alexander Hofmann. 2018. Evaluation of Microservice Architectures: A Metric and Tool-Based Approach. In *Information Systems in the Big Data Era (Lecture Notes in Business Information Processing)*, Jan Mendling and Haralambos Mouratidis (Eds.). Springer International Publishing, Cham, 74–89. https://doi.org/10.1007/978-3-319-92901-9_8

[6] Marcus Frisell. 2018. *Information visualization of microservice architecture relations and system monitoring : A case study on the microservices of a digital rights management company - an observability perspective.* Master's thesis. Royal Institute of Technology, Stockholm, Sweden. http://www.diva-portal.org/smash/record.jsf?pid=diva2:1240044&dswid=-8254

[7] Giona Granchelli, Mario Cardarelli, Paolo Di Francesco, Ivano Malavolta, Ludovico Iovino, and Amleto Di Salle. 2017. MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-Based Systems. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 298–302. https://doi.org/10.1109/ICSAW.2017.9

[8] Danny Holten and Jarke J. van Wijk. 2009. A user study on visualizing directed edges in graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. Association for Computing Machinery, New York, NY, USA, 2299–2308. https://doi.org/10.1145/1518701.1519054

[9] Martin Kleehaus, Ömer Uludağ, Patrick Schäfer, and Florian Matthes. 2018. MICROLYZE: A Framework for Recovering the Software Architecture in Microservice-Based Environments. In *Information Systems in the Big Data Era*, Jan Mendling and Haralambos Mouratidis (Eds.). Springer International Publishing, Cham, 148–162. https://doi.org/10.1007/978-3-319-92901-9_14

[10] Oscar Manglaras, Alex Farkas, Peter Fule, Christoph Treude, and Markus Wagner. 2023. Problems in Microservice Development: Supporting Visualisation. In *2023 IEEE Working Conference on Software Visualization (VISSOFT)*. 62–72. https://doi.org/10.1109/VISSOFT60811.2023.00017 ISSN: 2832-6555.

[11] Benjamin Mayer and Rainer Weinreich. 2017. A Dashboard for Microservice Monitoring and Management. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. 66–69. https://doi.org/10.1109/ICSAW.2017.44

[12] Tamara Munzner. 2014. *Visualization Analysis and Design* (1st edition ed.). A K Peters/CRC Press, Boca Raton.

[13] Pattarakrit Rattanukul, Chansida Makaranond, Pumipat Watanakulcharus, Chaiyong Ragkhitwetsagul, Tanapol Nearunchorn, Vasaka Visoottiviseth, Morakot Choetkiertikul, and Thanwadee Sunetnanta. 2023. Microusity: A testing tool for Backends for Frontends (BFF) Microservice Systems. *2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC)* (2023), 74–78. https://doi.org/10.1109/ICPC58990.2023.00021

[14] Colin Ware. 2013. *Information visualization: perception for design, third edition* (3rd ed. ed.). Morgan Kaufmann, Waltham, Mass.

[15] Tom Černý, Amr Abdelfattah, Vincent Bushong, Abdullah Maruf, and Davide Taibi. 2022. Microvision: Static analysis-based approach to visualizing microservices in augmented reality. In *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. 49–58. https://doi.org/10.1109/SOSE55356.2022.00012

Received 2024-01-29; accepted 2024-04-15