Smart and Adaptive Routing Architecture: An Internet-of-Things Traffic Manager Based on Artificial Neural Networks

Amirali Amiri
University of Vienna, Software Architecture Group
University of Vienna, Doctoral School Computer Science
Vienna, Austria
amirali.amiri@univie.ac.at

Uwe Zdun
University of Vienna
Software Architecture Group
Vienna, Austria
uwe.zdun@univie.ac.at

***Abstract*—Many studies have been performed on integrating the Internet of Things (IoT) with cloud services. As these systems become widely used, quality metrics are of concern. For example, users might specify access control to restrict their sensitive data being processed in the cloud. Routers, e.g., API gateways, message brokers, or sidecars, can provide this access control by blocking or routing device data to a specific cloud service. However, a static routing application might not suit the dynamic behavior of IoT applications well. For example, in a centralized schema, where all device data is routed to a component for control checking, performance can be an issue. On the other hand, distributed routing can harm the reliability of a system, as device data might be lost due to an unresponsive service. We present the Smart and Adaptive Routing (SAR) architecture that creates an optimal reconfiguration solution using a deep neural network based on the quality metrics of an IoT application. To design our architecture, we give a background of the published studies and a review of the gray literature, e.g., practitioner blogs, to categorize the knowledge in the domain of IoT-cloud traffic management. We systematically evaluate our approach in an extensive evaluation of 4500 cases and compare SAR with an empirical data set of 1200 hours. The results show that our approach significantly improves quality-of-service measures by adapting the IoT-cloud system at runtime.**

***Index Terms*—Self-Adaptive Systems, Dynamic Routing Architectures, Internet of Things, Reliability and Performance Trade-Offs, IoT-Cloud Traffic Management, Artificial Neural Networks**

I. Introduction

The integration of IoT and cloud technologies is becoming increasingly popular as it enables the analysis of vast amounts of data and facilitates the development of innovative applications. Various authors have attempted to document effective approaches and recurring patterns in the intersection of IoT and cloud technologies. However, established practices within the industry are also available in the "gray literature" such as practitioner blogs, experience reports, and system documentation. Without a proper knowledge categorization, these sources provide limited insights into the existing practices as they vary and rely heavily on personal experience. An architect must have substantial experience or study a comprehensive set of knowledge sources to understand the state-of-the-art.

Even for an expert architect, it is hard to consider the multifaceted design of IoT-cloud integration applications without supporting tools and frameworks. Assume there is a privacy requirement for an application to restrict the IoT device data access to specific cloud services, e.g., as imposed by the general data protection regulation[1]. In such a case, dynamic routers in the cloud, e.g., API gateways [33], message brokers [19] or sidecars [16], [21], [26] can route or block device data based on specific rules.

Considering the dynamic nature of today's IoT systems, a static application, e.g., centralized routing, where all IoT data is routed to one component for control checking, might not be suitable. In this example, centralized routing might introduce additional latency harming the system's performance. On the other hand, distributed routing might decrease reliability requirements, as there might be data loss due to an unavailable service. A system adaptation at runtime based on the quality-of-service metrics can help ensure IoT quality requirements. This paper aims to study the following research questions:

RQ1: *What are the patterns and best practices that support IoT-cloud traffic management, and how are they related to each other?*

RQ2: *What is the architecture of an automation framework that analyses an IoT system at runtime and adapts the system's configuration based on quality-of-service metrics?*

RQ3: *How do the quality-of-service predictions of the chosen optimal solution compare with the case where one architecture runs statically, i.e., without adaptation?*

The contributions of our research are as follows: Firstly, we present a knowledge categorization within the IoT-cloud integration domain and dynamic routing of device data by summarizing the findings of published studies and a gray literature review. We categorize the patterns and best practices into three widely-used architectural patterns. Secondly, we introduce the Smart and Adaptive Routing (SAR) architecture,

[1]https://gdpr.eu

which is a self-adaptive architecture based on the Monitor, Analyze, Plan, Execute, Knowledge (MAPE-K) loops [6], [7], [20]. Using an Artificial Neural Network (ANN) [28], SAR produces an optimal reconfiguration solution for the routing schema. Thirdly, we provide a novel data set for training the ANN that consists of 36336300 data points. This data set considers different runtime configurations and outputs a multi-criteria optimization [3] of reliability and performance. We provide a prototypical tool to use the ANN and our concepts.

We evaluate our approach by comparing the SAR reliability and performance predictions with our empirical data set of 1200 hours in an extensive evaluation of 4500 cases. Our systematic evaluation shows that SAR leads to significant increases in reliability and performance in cases where the wrong architecture is chosen. Even on average, when correct and incorrect architecture choices are analyzed, SAR gives 13.53% reliability and 28.55% performance improvements.

The structure of the paper is as follows: Section II presents the overview of our approach. Section III categorizes the knowledge in the domain of IoT-cloud traffic management. Section IV gives the details of the SAR architecture. Section V presents the evaluation, and Section VI discusses the threats to the validity of our research. We study the related work in Section VII and conclude in Section VIII.

II. Approach Overview

First, we study the established practices in IoT-cloud traffic management. We do so by giving a background on the published literature, and a qualitative grey literature review [17], [18], e.g., practitioner reports, system documentation, and technical blogs. Our gray literature review is based on the Grounded Theory (GT) [12], [13] to formalize current practitioners' understanding and architectural concepts of IoT-cloud traffic management. Then, we categorize the traffic management best practices into three commonly-used routing patterns from a high level of abstraction. Having done this categorization, we introduce the SAR architecture that automatically adapts between the categorized routing patterns based on the monitored runtime data. We use an ANN [28] for regression analysis, i.e., deep learning of our training dataset. Using these machine-learning techniques, we reduce response latency and resource costs of our architecture. Moreover, the ANN allows our architecture to predict measurements for the runtime data not included in our dataset.

The GT research approach is a methodical process of deriving theory from data. Throughout the interpretation of data, the objective is to construct a theory firmly rooted in the collected data. Data analysis should occur concurrently with data collection rather than after the fact. Constant comparison is the primary activity in grounded theory, where the researcher continually compares new data with pre-existing data and concepts. Any newly-emerging abstract concepts should also be compared with pre-existing information. These concepts are then organized into categories and linked through relations. The concepts, categories, and properties derived from the data should guide subsequent research activities. Theoretical sampling involves actively seeking out new data based on the outcomes of the previous iteration and determining what kinds of data should be collected next [22]. This process continues until Theoretical Saturation is reached, i.e., "the point in category development at which no new properties, dimensions, or relationships emerge during analysis" [13].

III. Routing Architecture Patterns

In this section, we present a background of the published studies and a review of gray literature to categorize the knowledge in the IoT-cloud traffic management into three routing architecture patterns. Note that we limit the scope of our study to cloud-integrated IoT devices, e.g., using digital twins [23], and study the cloud-based routing of the traffic generated by such devices. We assume that the data of IoT devices are processed using the integration platforms, e.g., normalized, filtered, and aggregated. Defining and focusing on a scope provides useful insight as the IoT devices and platforms are vastly different.

A. Background on Published Studies

There are many studies in the literature on traffic routing. On the one hand, some studies take a centralized approach where all traffic is processed and routed centrally, e.g., using API Gateways [33], event stores as well as event streaming platforms [33], Message Brokers [19], or any kind of central service bus, e.g., Enterprise Service Buses [11]. One benefit of this architecture is that it is easy to manage, understand, and change as all control logic regarding request flows is implemented in one component. However, this introduces the drawback that the design of the internals of the central entity component is a complex task. Another advantage is that in an application, which consists of stateful request flow sequences, the state does not need to be passed between various distributed components. Nonetheless, services need to call back to the central entity component to fetch the saved state of prior stages to proceed with the next step in the request flow sequence.

On the other hand, some works use distributed approaches, such as sidecars [16], [21], [26] that attach to services and rout the incoming calls. In contrast to the central routing approaches, the traffic management is distributed. Sidecars offer a separation of concerns since the control logic regarding request flow is implemented in a different component than the service. However, they are tightly coupled with their directly linked services. Sidecars offer benefits whenever decisions need to be made structurally close to the service logic. One advantage of this approach is that, compared to central routing, it is usually easier to implement sidecars since they require less complex logic to control the request flow of their connected services. However, it is not always possible to add sidecars, e.g., when services are off-the-shelf products.

Typically, an application uses a combination of central and distributed routing schemas based on its need. For example, consider an API Gateway, two event streaming platforms,

TABLE I: The Investigated Gray Literature Cases

ID	Title	URL
S1	Understand the Azure IoT Edge runtime and its architecture	https://docs.microsoft.com/en-us/azure/iot-edge/iot-edge-runtime?view=iotedge-2020-11
S2	Connecting IoT devices to the cloud	https://www.thoughtworks.com/insights/blog/iot/connecting-iot-devices-cloud
S3	Real-time Data Streaming in IoT: Why and How	https://solace.com/blog/real-time-data-streaming-in-iot
S4	Edge to Twin: A scalable edge to cloud architecture for digital twins	https://aws.amazon.com/de/blogs/iot/edge-to-twin-a-scalable-edge-to-cloud-architecture-for-digital-twins
S5	Understanding edge computing for manufacturing	https://www.redhat.com/en/topics/edge-computing/manufacturing
S6	How to use Digital Twins for IoT Device Configurations	https://tributech.io/blog/digital-twins-for-IoT-device-configurations
S7	Understand and use device twins in IoT Hub	https://learn.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-device-twins
S8	Mainflux 0.11 — Digital Twin, MQTT Proxy And More	https://medium.com/mainflux-iot-platform/mainflux-0-11-digital-twin-mqtt-proxy-and-more-46bde98635fe
S9	Connecting OPC UA Publisher to Amazon AWS IoT with MQTT	https://www.prosysopc.com/blog/aws-iot-mqtt-demo
S10	Dataworks: Internet Of Things	https://www.dataworks.ie/iot-a-step-by-step-guide-on-how-to-connect-devices-to-the-cloud
S11	Cloud Computing for the Internet of Things (IoT)	https://dgtlinfra.com/cloud-internet-of-things-iot
S12	IoT Gateway User Guide	https://docs.devicewise.com/Content/Products/GatewayDevelopersGuide/IoT-Gateway-User-Guide.htm
S13	Configuring Envoy as an edge proxy	https://www.envoyproxy.io/docs/envoy/latest/configuration/best_practices/edge
S14	Google Vulnerability Reward Program (VRP)	https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/security/google_vrp
S15	Control your traffic at the edge with Cloudflare	https://blog.cloudflare.com/cloudflare-traffic
S16	The Distributed System ToolKit: Patterns for Composite Containers	https://kubernetes.io/blog/2015/06/the-distributed-system-toolkit-patterns
S17	Kong Embedded: A New Way of Deploying Kong Enterprise on Edge Devices	https://konghq.com/blog/kong-embedded-a-new-way-of-deploying-kong-enterprise-on-edge-devices
S18	NGINX Plus for the IoT: Load Balancing MQTT	https://www.nginx.com/blog/nginx-plus-iot-load-balancing-mqtt
S19	Using NGINX with IoT: Ingress to the Edge and Beyond	https://www.nginx.com/resources/videos/using-nginx-with-iot-ingress-to-the-edge-and-beyond
S20	Cost-effective cloud edge traffic engineering with Cascara	http://cascara.network

and several sidecars, all making routing decisions in the application. We categorize the traffic manager components in these studies under dynamic routers [19]. The benefit of this architecture is that routers can use local information regarding request routing amongst their connected services. For instance, if a set of services are dependent on one another as steps of processing a request, routers can be used to facilitate dynamic routing. Nonetheless, these routers introduce an implementation overhead regarding data structures, control logic, management, deployment, and so on since they are usually distributed on multiple hosts.

We review the gray literature to study the IoT-cloud integration and the currently-used approaches.

B. Gray Literature Review

Grey literature in software engineering is “any material about SE that is not formally peer-reviewed nor formally published” [17], e.g., blog posts, articles, presentations, and audio-video material [18]. Grey literature is chosen in addition to the published literature because we wish to include an understanding of practitioners’ views within this domain in our research. Such data sources are most representative of these views. Rainer and Williams [32] describe various benefits to grey literature sources in software engineering research, including that these sources “promote the voice of the practitioner” and “provide information on practitioners’ contemporary perspectives on important topics relevant to practice and to research.”

To begin the search for a few appropriate and detailed sources from the grey literature, we rely on the authors’ accounts of established practices. These sources, which are selected for their suitability, are subsequently employed as impartial depictions of practices during subsequent analysis. Table I lists the gray literature cases investigated in this study.

C. Knowledge Categorization

Figure 1 presents the categorization of knowledge in the domain of IoT-cloud traffic management. The root of our model is a *Traffic Manager*. We categorized the centralized and distributed routing using a *Dynamic Router*. These routers include *Sidecar* and *API Gateway*. Different patterns provide gateway functionalities, i.e., *Enterprise Service Bus*, *Message Gateway*, *Event Streaming*, and *Proxy Ambassador*. Typical features of IoT-cloud applications, e.g., security, rate limiting, load balancing, health monitoring, and data analysis, are performed in the categorized dynamic routers.

As mentioned, we consider cloud-integrated IoT devices in this paper. Therefore, the *Traffic Manager* uses an *IoT-Cloud Integration* to communicate with these devices. A *Digital Twin* can use *Platform Integration* or *Data Steaming Integration* to update the *Device Metadata*, *Device Data*, *Device Visualization*, and *Device Control/Configuration*. Typical features regarding data of IoT devices, e.g., gathering, normalization, filtering, aggregation, and anomaly detection, are performed. *IoT-Cloud Integration* uses *IoT-Cloud Communication* that involves IoT devices sending data to cloud services for storage, processing, and analysis. There are various communication patterns and practices.

We categorized these patterns into *Synchronous Call*, *Data Streaming*, and *Asynchronous Call*. *Synchronous Call* offers a straightforward means of exchanging data between IoT and the cloud where the device blocks and waits for the response. The pattern *Data Streaming* refers to a continuous

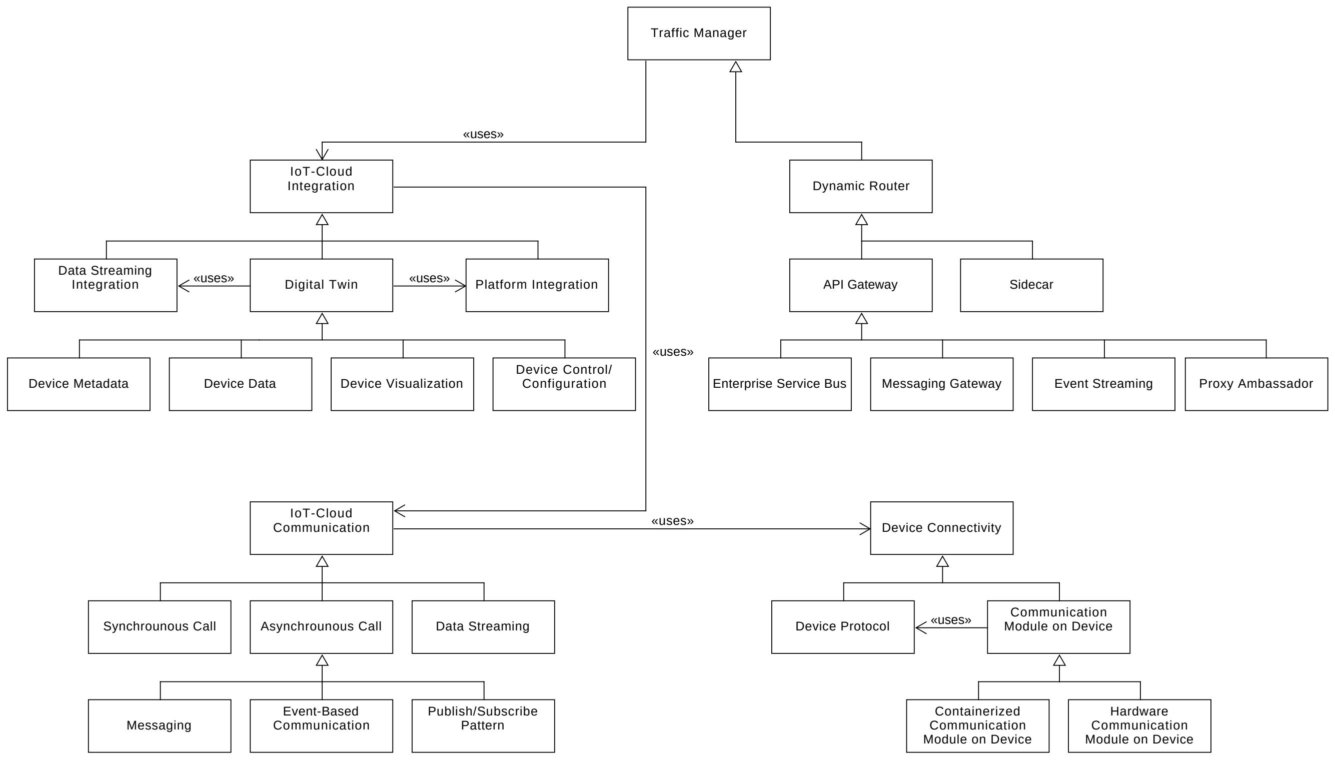


Fig. 1: Knowledge Categorization of IoT-Cloud Traffic Management

flow of data that is generated, transmitted, and received in real-time. The *Asynchronous Call* implements the request/response asynchronously. *Messaging* is a lightweight transport protocol for connecting remote devices. *Publish/Subscribe Pattern* uses topics to identify messages and route them to publishing and subscribing clients. *Event-based interaction* ensures that all changes to the state are stored as a sequence of events. If a system utilizes the patterns associated with asynchronous communication, it can lead to interactions that are more loosely coupled, eliminating the need for point-to-point interactions.

Device Connectivity is used for IoT devices to send data. A *Communication Module on Device* can be a *Hardware Communication Module on Device*. Since we study cloud-integrated devices, a *Containerized Communication Module on Device* is the focus of our study. These modules use *Device Protocol* to gather and transmit data. Note that we do not consider scenarios where devices communicate directly using peer-to-peer protocols[2].

D. Routing Patterns

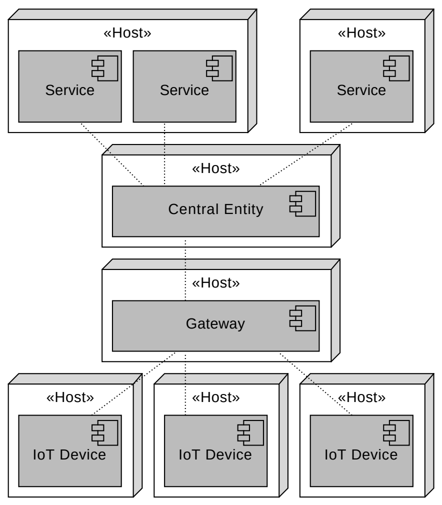
Based on the categorized knowledge in the previous section, we summarize the routing patterns into three patterns, *central routing*, *sidecar-based routing*, and the *dynamic routers*. The schematic of the *central routing* is shown in Figure 2a. Gateway is the entry point of the application where data from IoT devices are gathered. The central entity routes the data to appropriate services and checks for security, privacy, or conformance to any rules. Typically the gateway and the central entity are combined, e.g., when using an API gateway [33]. *Sidecar-based routing* is shown in Figure 2b. A sidecar [16], [21], [26] is attached to each service and makes routing decisions at the point of entry. For example, the sidecar checks for permissions if the service can process the received device data. The schematic of the *dynamic routers* [19] is shown in Figure 2c. Any combination of the above two routing patterns is represented by the *dynamic routers*.

IV. Architecture Details

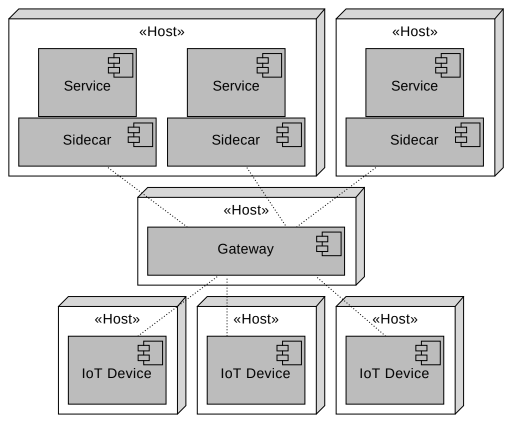
We propose a new approach that realizes all three architecture patterns shown in Figure 2. We hypothesize that a dynamic self-adaptation between the three architecture patterns is beneficial over any fixed selections. If a traffic and load change occurs, our approach can dynamically self-adapt the degrees to which more or less central routing is used to optimize its impact on quality-of-service measures, e.g., performance and reliability, trade-offs. These are important quality measures because, on the one hand, a central routing of IoT data can harm performance. On the other hand, distributed routing can decrease reliability since device data can be lost due to unavailability of services.

We design the Smart and Adaptive Routing (SAR) architecture that uses an artificial neural network to adapt between the routing architecture patterns automatically. SAR is based on Monitor, Analyze, Plan, Execute, Knowledge (MAPE-

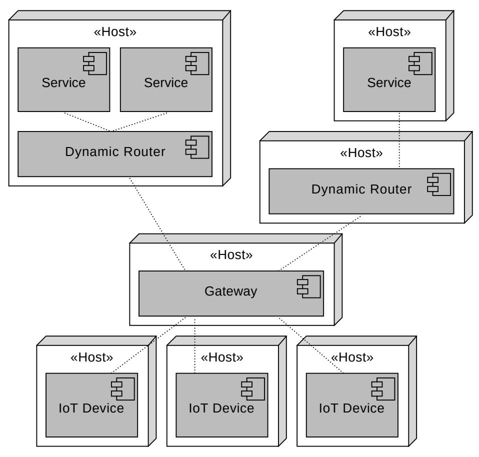
[2]see, e.g., https://husarnet.com/iot



(a) Central Routing



(b) Sidecar-Based Routing



(c) Dynamic Routers

Fig. 2: Routing Architecture Patterns

K) loops [6], [7], [20] and dynamically adapts between the architecture patterns on-the-fly. We define a concept called *router* and abstract all the controlling logic components, i.e., the central entity servicec (e.g., an API gateway), the dynamic routers, and the sidecars, under the router. This high-level router abstraction can be used to reconfigure the routing architecture dynamically. That is, we can change between the three architecture patterns moving from a centralized approach with one router to a distributed system with more routers (or vice versa) to adapt based on the need of an IoT application.

A. Metamodel

Figure 3 presents the metamodel of our architecture. A *Model* describes multiple *Hosts* and *Components*. Each *Component* is deployed on (up to) one *Host* at each point in time, which is any execution environment for these components, either physical or virtual, including cloud-integrated IoT devices such as digital twins [23]. *Request* models the request flow, linking a source and a destination component. There are several different component types. *IoT Devices* send *Device Data* to *Gateways*. To process these requests, *Gateways* send *Internal Requests* to *Routers* and *Services*. *Routers* and *Services* are both *Reconfigurable Components*, i.e., they are the adaptation targets. The *Configurator Components* perform the reconfiguration. *Monitor* observes *Reconfigurable Components* and the requests that pass the *Gateways*. *Manager* manages the reconfiguration and the *Scheduler* reschedules the containers.

B. Example of a Routing Configuration

Figure 4 presents a component diagram of a sample configuration, in which dashed lines represent the data flow and solid lines the reconfiguration control flow of an application. As shown, IoT devices access the system via a gateway that publishes monitoring data to the Quality of Service (QoS) monitor. The configuration manager observes the monitoring data and triggers a reconfiguration. To do so, the manager triggers the scheduler to reschedule the containers. Moreover, the manager calls the visualizer component to visualize the reconfiguration using PlantUML[3].

C. Reconfiguration Activities of the Dynamic Configurator

Figure 5 shows the reconfiguration activities of the dynamic configurator. The QoS monitor reads monitoring data and checks for reconfiguration, e.g., when metrics degradations are observed. Moreover, the reconfiguration can be triggered periodically or manually by an architect. When a reconfiguration is triggered, the reconfiguration manager consumes the monitoring data, and chooses a final reconfiguration solution using an artificial neural network (see Section IV-D for details). Based on this analysis, the scheduler reschedules the containers and visualizes the reconfiguration. As mentioned, our architecture is based on MAPE-K loops [6], [7], [20]. The QoS monitor implements the *monitor* and *analyze* stages, the manager develops the *plan*, and the scheduler realizes the *execute* step. We use our knowledge in Figure 1 as *knowledge*.

[3]https://plantuml.com/

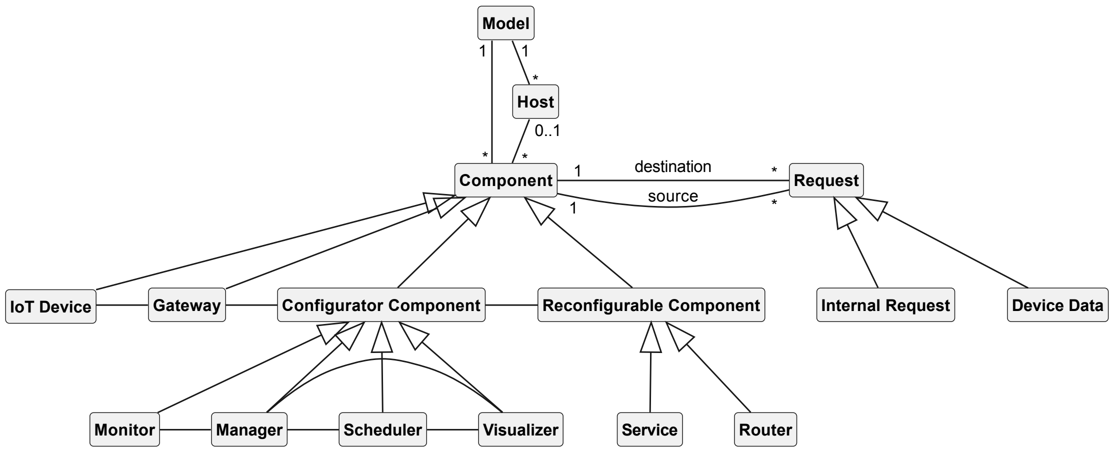


Fig. 3: Metamodel of the SAR Architecture

D. Artificial Neural Network

The manager component uses an ANN that predicts a reconfiguration solution, i.e., the number of routers in an application, based on the monitoring data. We use the TensorFlow[4] and the Keras sequential model[5] to create a deep neural network [28]. A sequential model has exactly one

[4]https://www.tensorflow.org/
[5]https://www.tensorflow.org/guide/keras/sequential_model

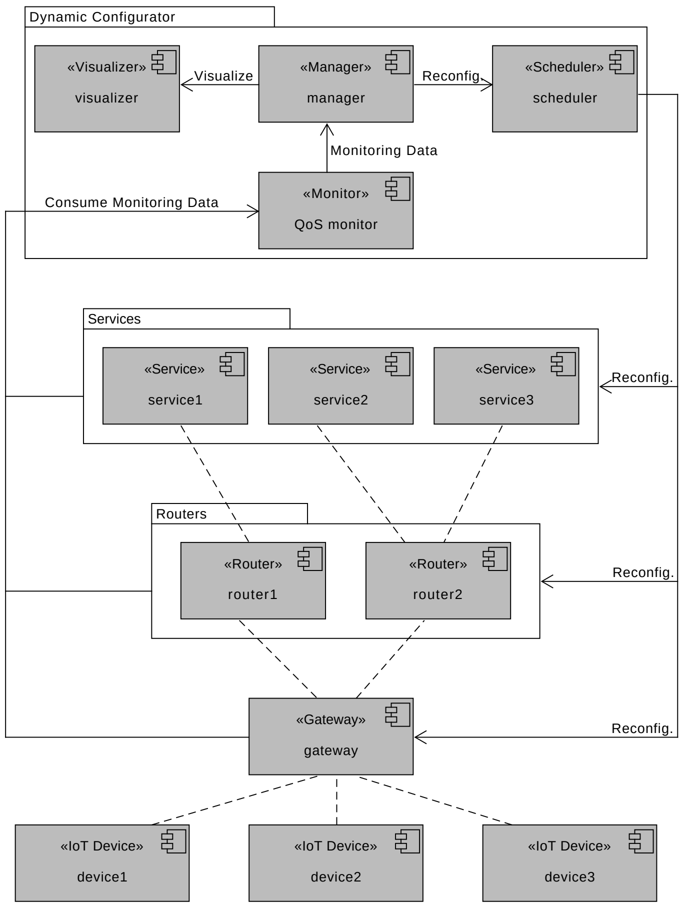


Fig. 4: Component Diagram of an Example Configuration
(The dashed represent the data flow and the solid lines the reconfiguration control flow of an application.)

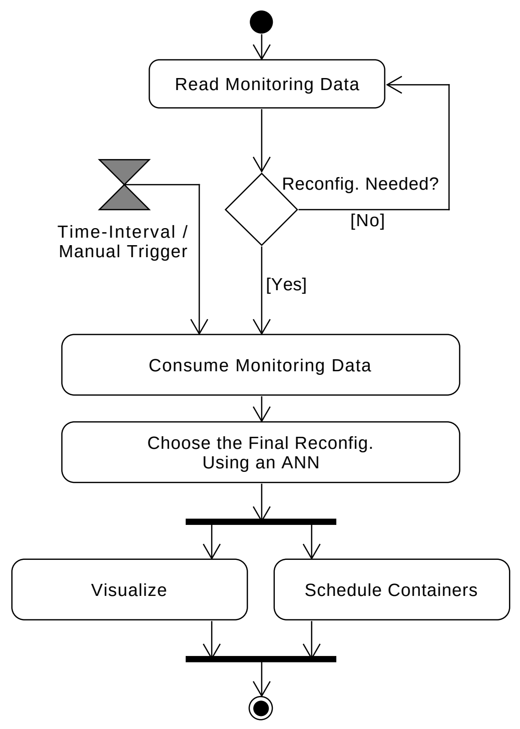


Fig. 5: Activities of the Dynamic Configurator

input and one output. We give a six-dimensional array of the monitoring data as input (see details below). The output is the reconfiguration solution based on the number of routers. One router indicates the central routing, and more routers specify distributed routing (see Section III-D). Our sequential model has 5 densely-connected layers, each with 40 neurons. We use the standard rectifier linear unit activation function, the mean squared error loss function, and the Adam optimizer [9] with a learning rate of 0.001.

One contribution of this study is a data set with 36336300 data points that we use for the training of the deep neural network. In our prior work [5], we performed a Multi-Criteria Optimization (MCO) analysis [3] to adapt the reliability and performance trade-offs of dynamic routing applications. We observed the incoming call frequency and the current architecture configuration, i.e., the number of services and routers in a system (see Figure 4). When metrics degradation occurred, our approach reconfigured the routing schema, that is, more centralized or distributed routing, to adjust the trade-offs. Architects give reliability and performance thresholds and weights to find the final reconfiguration solution based on the number of routers. The input of our training data set is all combinations of the following six monitoring data, and the output is the result of the MCO analysis for each case.

Let cf be the call frequency, n_{serv} the number of services, n_{rout} the number of routers, R_{th} the reliability threshold based on the number of request loss per second, P_{th} the performance threshold as the average processing time of requests per router, and PW the performance weight, i.e., a number between 0 and 1 giving the importance of performance compared to reliability. The importance weights are required to choose a final reconfiguration solution based on the need of different applications. For example, giving a higher weight to performance opts for more distributed routing because of the parallel processing of requests. We have the following ranges in our data set (see below for rationale):

$$3 \leq n_{serv} \leq 10 \tag{1}$$

$$1 \leq n_{rout} \leq n_{serv} \tag{2}$$

$$10 \leq cf \leq 100\, r/s \tag{3}$$

$$1.1 \leq R_{th} \leq 2\, r/s \tag{4}$$

$$35 \leq P_{th} \leq 100\, ms \tag{5}$$

$$0.0 \leq PW \leq 1.0 \tag{6}$$

These values are based on an extensive experiment of 1200 hours in that we measured the quality-of-service metrics of dynamic routing applications. Our training data set can be downloaded in the online artifact of this paper to support reproducibility[6]. The call frequency of $cf = 100\, r/s$ (or even lower numbers) is chosen in many studies (see, e.g., [14], [37]). Therefore, we chose different portions between 10 to 100 r/s. As for the number of services n_{serv}, based on our experience and a survey on existing cloud applications in the literature and industry [4], the number of cloud services that are directly dependent on each other in a call sequence is usually rather low. As a result, we study 3 to 10 services in a call sequence. For the reliability and performance thresholds, we studied our empirical data and chose the worst-case scenarios for centralized and distributed routing. A performance weight of 1.0 emphasizes performance, and 0.0 gives weight to reliability when choosing the final solution.

E. Tool Support

We developed a prototypical tool to demonstrate the SAR architecture. The tool is available in our online artifact. Figure 6 shows the tool architecture. Our tool generates artifacts in the form of Bash[7] scripts and configuration files, e.g., infrastructure configuration data. These scripts can schedule containers using the Docker technology[8]. Additionally, we provide visualizations by generating diagrams in PlantUML[3] to study different scenarios. The frontend of our application provides the functionalities of the QoS monitor, i.e., to specify architecture configurations as well as model elements such as reliability and performance thresholds. This information is sent to the manager component in the backend that finds the final reconfiguration solution using a deep ANN. The manager sends this solution to the scheduler to generate deployment artifacts. A visualization is created in the backend and shown

[6] Published as an open-access artifact: https://zenodo.org/record/7771540
[7] https://www.gnu.org/software/bash/
[8] https://www.docker.com/

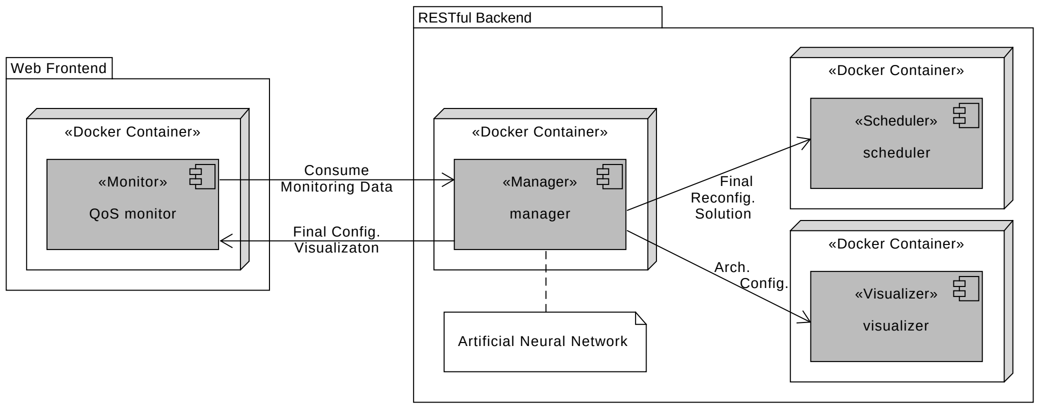


Fig. 6: Tool Architecture Diagram

in the frontend. The frontend is implemented in React[9] and the backend is developed in Node.js[10] and Python Flask[11].

V. Evaluation

In this section, we evaluate our architecture by comparing the ANN model predictions for the SAR adaptations to the fixed architecture patterns, i.e., central routing, sidecar-based routing, and dynamic routers (see Section III-D). We use the empirical results of our experiment, where we measured the reliability and performance metrics of the fixed architecture patterns. Note that SAR is neither specific to our experiment infrastructure nor to our cases. We use our empirical data (already reported in [4]) to evaluate our architecture.

A. Evaluation Cases

We systematically evaluate our proposed architecture through various architecture configurations, call frequencies, thresholds, and importance weights for reliability and performance. In our empirical data set, the routing patterns are measured with the following experiment 36 cases:

$$n_{serv} \in \{ 3, 5, 10 \} \tag{7}$$

$$n_{rout} \in \{ 1, 3, n_{serv} \} \tag{8}$$

$$cf \in \{ 10, 25, 50, 100 \} \; r/s \tag{9}$$

For reliability and performance thresholds and weights, we start from the lower bounds of our training data set (see Section IV-D) and study increments of 20% (5 levels for each):

$$1.1 \leq R_{th} \leq 2 \; r/s \tag{10}$$

$$35 \leq P_{th} \leq 100 \, ms \tag{11}$$

$$0.0 \leq PW \leq 1.0 \tag{12}$$

Overall we evaluate 4500 systematic evaluation cases: 36 experiment cases, 5 importance weights, 5 reliability thresholds, and 5 performance threshold levels.

[9]https://reactjs.org/
[10]https://nodejs.org/
[11]https://flask.palletsprojects.com/en/2.2.x/

B. Results Analysis

We define reliability gain, i.e., $RGain$, and performance gain, i.e., $PGain$, as the average percentage differences of our predictions compared to those of fixed architectures. These formulas are based on the Mean Absolute Percentage Error (MAPE) widely used in the cloud QoS research [38]. Let $R_{n_{rout}}$ and $P_{n_{rout}}$ reliability and performance predictions:

$$RGain = \frac{100\%}{n} \cdot \sum_{c \in Cases} \frac{R_c - R_{n_{rout}}}{R_{n_{rout}}} \tag{13}$$

$$PGain = \frac{100\%}{n} \cdot \sum_{c \in Cases} \frac{P_c - P_{n_{rout}}}{P_{n_{rout}}} \tag{14}$$

$Cases$ are our experiment cases; therefore, $n = 36$.

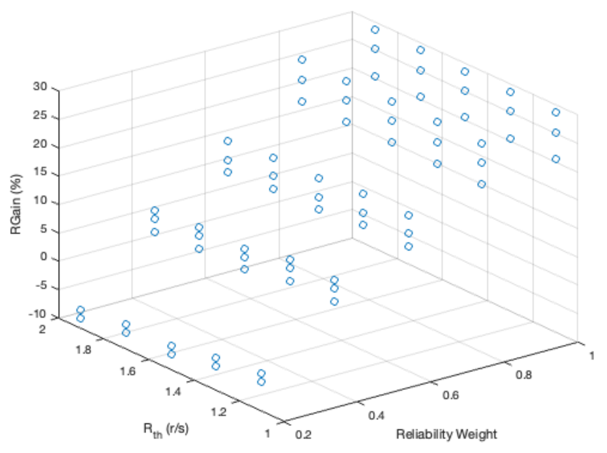
Figure 7 shows the reliability and performance gains compared to the predictions of fixed architecture configurations, i.e., without adaptations. Our adaptive architecture provides improvements in both reliability and performance gains. As more importance is given to the reliability of a system, i.e., reliability weight increases, our architecture reconfigures the routers so that the gain in reliability rises as shown by Figure 7a. Regarding performance, the same trend can be seen in Figure 7b. A higher performance weight results in a higher performance gain. On average, when cases with correct and incorrect architecture choices are analyzed together, our adaptive architecture provides 13.53% and 28.55% reliability and performance gains, respectively.

VI. Threats to Validity

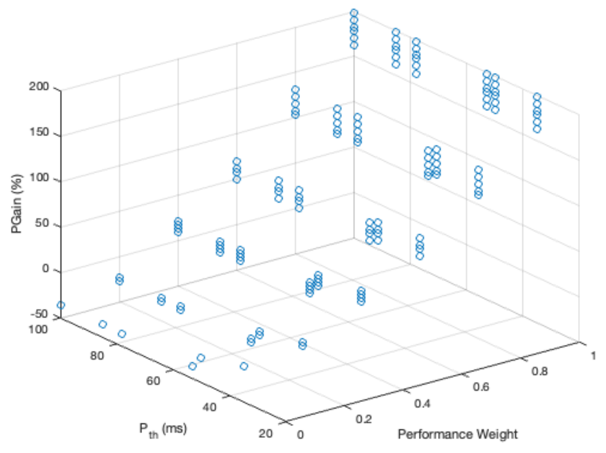
There are several threats to the validity of our study that we discuss based on the four threat types by Wohlin et al. [41].

A. Construct Validity

The accurate representation of the intended construct by a measurement is assessed through construct validity. In this paper, we studied cloud-integrated IoT devices, e.g., for creating digital twins [23]. The threat remains that other metrics might affect the results, e.g., latency introduced by communicating to the IoT device and waiting for a response. Moreover, there



(a) Reliability Gain



(b) Performance Gain

Fig. 7: Reliability and Performance Gains Compared to Fixed Architecture Configurations (Each point is an average of 36 evaluation cases)

are scenarios that devices directly communicate with each other using a peer-to-peer communication[2]. The IoT domain includes many different devices and platforms that one study cannot cover all aspects of the domain. Researchers have to define and focus on a specific scope to provide more useful insight. More research, probably with real-world devices, is required for this threat to be excluded.

B. Internal Validity

Internal validity concerns factors that affect the independent variables concerning causality. To increase internal validity, we reviewed the gray literature, e.g., practitioner reports produced independently of our study. This avoids bias, for example, compared to interviews in which the practitioners would be aware that their answers might be used in a study. However, the gray literature review introduces the threat that some important information might be missing in the reports. We tried to mitigate this threat by looking at many sources and comparing them to the published literature to ensure our knowledge categorization includes all important concepts in the IoT-cloud integration domain from an architectural view.

Regarding our proposed SAR architecture, the dynamic routing architecture patterns (see Section III-D) are based on different technologies. Our adaptive architecture abstracts the controlling logic component in dynamic routing under a router concept to allow interoperability between these architectures. In a real-world system, changing between these technologies is not always an easy task, but it is not impossible either. In this paper, we provided a scientific proof-of-concept based on an experiment with the prototypical implementation of these technologies. The threat remains that changing between these technologies in a real-world application might have other impacts on reliability and performance, e.g., network latency increasing processing time.

C. External Validity

External validity concerns threats that limit the ability to generalize the results beyond the experiment. Since we surveyed published studies and reached saturation in the review of gray literature, our categorized knowledge can be generalized to many kinds of applications in the IoT-cloud integration domain. Moreover, we designed our novel SAR architecture with generality in mind. The threat remains that evaluating our approach on a different infrastructure may lead to different results. To mitigate this thread, we systematically evaluated the proposed architecture with 4500 evaluation cases using the data of our extensive experiment (see Section V). Moreover, we used an artificial neural network. We trained it on our empirical data so that the results can be generalizable beyond the given experiment cases of 10 to 100 r/s and call sequences of 3 to 10 services.

D. Conclusion Validity

Conclusion validity concerns factors that affect the ability to conclude the relations between treatments and study outcomes. As the statistical method to evaluate the accuracy of our model's predictions, we defined reliability and performance gains based on the Mean Absolute Percentage Error (MAPE) metric [38] as it is widely used and offers good interpretability in our research context.

VII. Related Work

This section presents the related work of our study.

A. Architecture-Based Modeling

Numerous approaches have been proposed that study architecture-based performance prediction. Spitznagel and Garlan [36] present a general architecture-based model for performance analysis based on queueing network theory. Petriu et al. [30] present an architecture-based performance analysis that builds layered queueing network performance models from a UML description of the high-level architecture of a system. The Palladio component model [8] allows component modeling with relevant factors for performance properties and contains a simulation framework for performance prediction.

Architecture-based MCO [3] builds on top of these prediction approaches and the application of architectural tactics to search for optimal architectural candidates. Example MCO approaches supporting reliability and performance are ArcheOpterix [2], PerOpteryx [10], and SQuAT [31]. Sharma and Trivedi [34] present an architecture-based unified hierarchical model for software reliability, performance, security, and cache behavior prediction. Like our study, those works focus on supporting architectural design or decision-making. In contrast to our work, they do not focus on specific kinds of architecture or architectural patterns. Our approach focuses on the dynamic routing of IoT-cloud applications.

Vandikas et al. [39] conducted a performance analysis of their Internet of Things (IoT) framework to evaluate its behavior under heavy load produced by different amounts of producers and consumers. The main purpose of the framework is to allow producers, such as sensors, to publish data streams to which multiple interested consumers, e.g., external applications, can subscribe. This publish-subscribe functionality is realized by a central message broker implemented with RabbitMQ. In contrast to our work, dynamic data routing is not considered in this article; moreover, the performance evaluation of the framework focuses only on a single machine deployment, which may have led to results that are not easily generalizable to cloud-based applications.

B. Machine-Learning Approaches

Nafreen et al. [29] study architecture-based reliability modeling by considering learning-enable components using fault-tolerant machine-learning approaches. Similarly, Kumar et al. [24] investigate machine-learning techniques to predict software reliability. They study 16 software life cycle databases empirically and evaluate their predictions using mean absolute error, root mean squared error, correlation coefficient and precisions. A related research is performed by Wang [40] where the author studies the reliability of cyber-physical systems using different machine-learning techniques. Didona et al. [15] combines analytical models with machine learning approaches to predict the performance of software systems. These studies are related to our paper in that they use machine learning techniques to predict QoS measurements. Unlike our study, they focus on one aspect, either reliability or performance, and do not study the trade-offs of these QoS metrics.

C. Studies on Performability

Performability considers the effects of structural changes in a system, e.g., when a service is unresponsive (impeded reliability), on the overall performance of the system [35]. This is related to our research as we study the trade-offs of reliability and performance. Ahamad and Ratneshwer [1] provide a review on the performability of Safety-Critical Systems (SCS). They study the available approaches, as well as the metrics to evaluate the performability of SCS. Moreover, they define performance and reliability challenges in studying the SCS. This study is related to our work because it presents state of the art in performability studies. However, it does not provide a generalizable framework that can be used in the dynamic routing of IoT systems.

Mo et al. [27] study the performability analysis of multi-state sliding window systems. Like our study, they propose an approach based on multivalued decision diagrams. They analyze this approach in multiple case studies. Lisnianski et al. [25] present a Markov multi-state model for large-scale, highly responsive distributed systems. They provide an analytical performability model and present a short-term analysis to prevent performance and reliability decreases. Unlike our research, none of the above works focus of the IoT-cloud traffic management specifically.

VIII. Conclusion and Future Work

In this paper, we set out to answer what the patterns and best practices that support IoT architectures are, and how they are related to each other (**RQ1**), what the architecture of an automation framework is that analyses an IoT system at runtime and adapts the system's reconfiguration using an optimal solution (**RQ2**), and how the reliability and performance predictions of the chosen optimal solution compare with the case where one architecture runs statically, i.e., without adaptation (**RQ3**).

For **RQ1**, we surveyed the published literature as well as the gray literature and provided knowledge categorization in the IoT-cloud integration domain. For **RQ2**, we introduced the Smart and Adaptive (SAR) architecture that analyzes different run-time inputs and produces an optimal reconfiguration solution using a deep neural network. For **RQ3**, we systematically evaluated our approach using 4500 evaluation cases based on the empirical data of our extensive experiment (see Section V). The results show that the proposed architecture can adapt the routing pattern in a running IoT system to optimize reliability and performance. Even on average, where cases with the right and the wrong architecture choices are analyzed together, our approach offers a 13.53% reliability gain and a 28.55% performance gain.

To the best of our knowledge, no architecture has been presented in the literature that dynamically adjusts reliability and performance trade-offs, specifically in IoT-cloud traffic management. The SAR architecture adapts based on triggers, e.g., change of incoming load frequency or degradation of monitoring data using an ANN. For our future work, we plan to extend our novel architecture to consider IoT devices more in-depth such as deploying artifacts directly on the devices and covering more aspects of IoT communications, e.g., using peer-to-peer protocols.

Acknowledgment

This work was supported by FFG (Austrian Research Promotion Agency) project MODIS, no. FO999895431.

References

[1] S. Ahamad and Ratneshwer. Some studies on performability analysis of safety critical systems. *Computer Science Review*, 39:100319, 2021.

[2] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya. Archeopterix: An extendable tool for architecture optimization of AADL models. In *ICSE 2009 Workshop on Model-Based Methodologies for Pervasive and Embedded Software, MOMPES 2009*, pages 61–71. IEEE, 2009.

[3] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya. Software architecture optimization methods: A systematic literature review. *IEEE Trans. Software Eng.*, 39(5):658–683, 2013.

[4] A. Amiri, U. Zdun, and A. van Hoorn. Modeling and empirical validation of reliability and performance trade-offs of dynamic routing in service- and cloud-based architectures. In *IEEE Transactions on Services Computing (TSC)*, 2021.

[5] A. Amiri, U. Zdun, A. van Hoorn, and S. Dustdar. Automatic adaptation of reliability and performance tradeoffs in service- and cloud-based dynamic routing architectures. In *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2021.

[6] P. Arcaini, E. Riccobene, and P. Scandurra. Modeling and analyzing mape-k feedback loops for self-adaptation. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 13–23. IEEE, 2015.

[7] P. Arcaini, E. Riccobene, and P. Scandurra. Formal design and verification of self-adaptive systems with decentralized control. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 11(4):1–35, 2017.

[8] S. Becker, H. Koziolek, and R. Reussner. Model-based performance prediction with the palladio component model. In *Proceedings of the 6th International Workshop on Software and Performance*, WOSP '07, page 54–65, New York, NY, USA, 2007. ACM.

[9] S. Bock and M. Weiß. A proof of local convergence for the adam optimizer. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[10] A. Busch, D. Fuchss, and A. Koziolek. Peropteryx: Automated improvement of software architectures. In *IEEE International Conference on Software Architecture ICSA Companion 2019*, pages 162–165. IEEE, 2019.

[11] D. A. Chappell. *Enterprise service bus*. O'Reilly, 2004.

[12] K. Charmaz. Constructing grounded theory: a practical guide through qualitative analysis. *Sage Publications*, 2006.

[13] J. Corbin and A. Strauss. Basics of qualitative research: Techniques and procedures for developing grounded theory. *Sage Publications*, 1998.

[14] D. J. Dean, H. Nguyen, P. Wang, and X. Gu. Perfcompass: Toward runtime performance anomaly fault localization for infrastructure-as-a-service clouds. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, 2014.

[15] D. Didona, F. Quaglia, P. Romano, and E. Torre. Enhancing performance prediction robustness by combining analytical modeling and machine learning. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, page 145–156, New York, NY, USA, 2015. Association for Computing Machinery.

[16] Envoy. Service mesh. *https://www.learnenvoy.io/articles/service-mesh.html*, 2019.

[17] V. Garousi, M. Felderer, M. V. Mäntylä, and A. Rainer. *Benefitting from the Grey Literature in Software Engineering Research*, pages 385–413. Springer International Publishing, Cham, 2020.

[18] V. Garousi, M. Felderer, and M. V. Mäntylä. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, 106:101–121, 2019.

[19] G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. Addison-Wesley, 2003.

[20] D. G. D. L. Iglesia and D. Weyns. Mape-k formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 10(3):1–31, 2015.

[21] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018.

[22] R. B. Johnson and L. Christensen. Educational research: Quantitative, qualitative, and mixed approaches. *Sage Publications*, 2019.

[23] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks. Characterising the digital twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology*, 29:36–52, 2020.

[24] P. Kumar and Y. Singh. An empirical study of software reliability prediction using machine learning techniques. *International Journal of System Assurance Engineering and Management*, 3(3):194–208, 2012.

[25] A. Lisnianski, E. Levit, and L. Teper. Short-term availability and performability analysis for a large-scale multi-state system based on robotic sensors. *Reliability Engineering and System Safety*, 205:107206, 2021.

[26] Microsoft. Sidecar pattern. https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar, 2010.

[27] Y. Mo, L. Xing, L. Zhang, and S. Cai. Performability analysis of multi-state sliding window systems. *Reliability Engineering and System Safety*, 202:107003, 2020.

[28] G. Montavon, W. Samek, and K.-R. Müller. Methods for interpreting and understanding deep neural networks. *Digital Signal Processing*, 73:1–15, 2018.

[29] M. Nafreen, S. Bhattacharya, and L. Fiondella. Architecture-based software reliability incorporating fault tolerant machine learning. In *2020 Annual Reliability and Maintainability Symposium (RAMS)*, pages 1–6, 2020.

[30] D. Petriu, C. Shousha, and A. Jalnapurkar. Architecture-based performance analysis applied to a telecommunication system. *IEEE Transactions on Software Engineering*, 26(11):1049–1065, 2000.

[31] A. Rago, S. A. Vidal, J. A. Diaz-Pace, S. Frank, and A. van Hoorn. Distributed quality-attribute optimization of software architectures. In *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2017*, pages 7:1–7:10. ACM, 2017.

[32] A. Rainer and A. Williams. Using blog-like documents to investigate software practice: Benefits, challenges, and research directions. *Journal of Software: Evolution and Process*, 31, 08 2019.

[33] C. Richardson. Microservice architecture patterns and best practices. *http://microservices.io/index.html*, 2019.

[34] V. S. Sharma and K. S. Trivedi. Architecture based analysis of performance, reliability and security of software systems. In *Proceedings of the 5th International Workshop on Software and Performance*, WOSP '05, page 217–227, New York, NY, USA, 2005. Association for Computing Machinery.

[35] R. Smith, K. Trivedi, and A. Ramesh. Performability analysis: measures, an algorithm, and a case study. *IEEE Transactions on Computers*, 37(4):406–417, 1988.

[36] B. Spitznagel and D. Garlan. Architecture-based performance analysis. In *Proc. the 1998 Conference on Software Engineering and Knowledge Engineering*. Carnegie Mellon University, June 1998.

[37] O. Sukwong, A. Sangpetch, and H. S. Kim. Sageshift: managing slas for highly consolidated cloud. In *2012 Proceedings IEEE INFOCOM*, pages 208–216, 2012.

[38] K. S. Trivedi and A. Bobbio. *Reliability and availability engineering: modeling, analysis, and applications*. Oxford University Press, 2017.

[39] K. Vandikas and V. Tsiatsis. Performance evaluation of an iot platform. In *Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on*, pages 141–146. IEEE, 2014.

[40] H. Wang. Research on real-time reliability evaluation of cps system based on machine learning. *Computer Communications*, 157:336–342, 2020.

[41] C. Wohlin, P. Runeson, M. Hoest, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering*. Springer, 2012.