Towards Extraction of Message-Based Communication in Mixed-Technology Architectures for Performance Model

Snigdha Singh, Yves Richard Kirschner, Anne Koziolek
{snigdha.singh,yves.kirschner,anne.koziolek}@kit.edu
Karlsruhe Institute of Technology
Germany

ABSTRACT

Software systems architected using multiple technologies are becoming popular. Many developers use these technologies as it offers high service quality which has often been optimized in terms of performance. In spite of the fact that performance is a key to the technology-mixed software applications, still there a little research on performance evaluation approaches explicitly considering the extraction of architecture for modelling and predicting performance.

In this paper, we discuss the opportunities and challenges in applying existing architecture extraction approaches to support model-driven performance prediction for technology-mixed software. Further, we discuss how it can be extended to support a message-based system. We describe how various technologies deriving the architecture can be transformed to create the performance model. In order to realise the work, we used a case study from the energy system domain as an running example to support our arguments and observations throughout the paper.

CCS CONCEPTS

• **Software engineering Architecture recovery**; • **Information systems Distributed system**; • **Applied computing Enterprise applications**; • **Service-oriented architectures**;

KEYWORDS

Architecture-level Performance Model, Component-based Software, Reverse Engineering, model-driven Engineering

ACM Reference Format:
Snigdha Singh, Yves Richard Kirschner, Anne Koziolek. 2021. Towards Extraction of Message-Based Communication in Mixed-Technology Architectures for Performance Model. In *ACM/SPEC International Conference on Performance Engineering Companion (ICPE '21 Companion), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3447545.3451201

ICPE '21 Companion, April 19-23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8331-8/21/04…$15.00
https://doi.org/10.1145/3447545.3451201

1 INTRODUCTION

Mixed-technology architecture, commonly abbreviated as "heterogeneous architecture" or "microservice architecture" is the combination of several technologies. Mixed-technology software system usually consists of: lightweight container technologies (LXC and Docker), service discovery technologies (Eureka and Consul, zookeeper), container orchestration technologies (Mesos and Kubernetes), continuous-delivery technologies (Ansible and Drone) and many more other technologies as required by the specific software [11]. Each of these technologies are individual components and use loosely coupled services running in their own processes. Components usually deployed in service containers like Docker and communicate with each other via lightweight communication protocols. Each component defines an interface that other components can consume. The services offered by the components, communicate via RESTful APIs (Representational State Transfer Application Program Interface) or message brokers in certain cases. This style facilitates the services for independent development, scaling, deployment, service discovery and load balancing without compromising the integrity of the application.

From both development and operation prospective such as, deploying, running, and monitoring enormous number of service instances pose a huge challenge. To address the development challenges, performance simulations are promising option. Performance modelling and prediction based on architecture, is well accepted in the community for design-time and run-time model extraction and performance evaluation. For performance model construction, it is essential to capture the component interaction as it has a great impact on performance. However, the nature of interaction makes it harder to extract component relations from static sources since component relationships may only be captured at run-time dynamically for such systems.

In order to collect the right set of performance parameters for performance model construction, it is important to analyze the communication relationships between services, which requires a monitoring infrastructure. This dynamic information can be combined with static information like API description and service to generate architecture-based performance prediction model [1, 3]. Several reverse engineering approaches used for this purpose to recover the up-to-date component-based architectural model from the current implementation of such mixed-technology system. But, most of the approaches fail to provide "sufficient and required" performance-related attribute to model and predict architecture based performance prediction for mixed-technology system. In our observation, the objective of achieving an automated architecture model from the system implementation mixed-technology system is challenged by certain points.

- in-sufficient performance model parameters
- unavailability of appropriate hybrid methods for mixed-technology architecture extraction
- difficulty in capturing the performance impact component interaction during run-time
- lack of proper formalisation method to extract the performance model from current implementation

In our work, we particularly investigate the above-mentioned challenges and discuss how our approach aim to tackle them. In our long term goal and research output we aim to provide an acceptable solution to the community to deal with such issues.

2 FOUNDATION

2.1 Component-Based Software

Component-based software engineering (CBSE) facilitates building a modular and reusable system [15]. This enables a faster and low cost development time. Due to the above-mentioned advantages, mixed-technology software adheres to the principle of CBSE and builds upon several components in a distributed software development environment.

A component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard [10]. CBSE is reinforced by component model. Component frameworks such as OSGi or EJB define components and how to compose them. Components are composed by connectors which connects components via their ports. Ports are responsible for interaction between the components. A component can have multiple interfaces (defined by set of ports) to communicate with each other. Components define their behavior by required and provided interfaces (ports). Services of components are encapsulated through their required and provided interfaces. In order to simulate the performance of the software system, it is vital to capture the communication between the components and their behavioural specification. Such parameters will then be provided as inputs to the performance model for performance prediction.

2.2 Model-Based Performance Prediction

The goal of performance model is to deliver an abstract representation of a real system that captures its performance properties i.e. run-time resource demanding, capacity planning and so on. Later these quantitative attributes are given to the model to measure the performance of the system as accurately as possible. Model-based approaches again divided into two types: analytical and architecture-level (architecture-based) performance models. Major factors influencing performance like system architecture, resources and usage-scenario are derived from architecture-level [5]. There exists two models, the Palladio Component Model (PCM) [19] and the Descartes Modeling Language (DML) [12], with a major focus to predict the performance of CBSE based on architecture-level model. PCM performance model either can be generated at design-time from the design specifications or can be extracted from the running application of the systems with the help of appropriate reverse engineering process.

2.3 Palladio Component Model

PCM is a modelling language used to model and simulate architecture-based performance prediction. PCM has been used to predict the software performance in various industries. Based on PCM model, it is possible to perform simulation to estimate the performance of the system. The Palladio approach also provides modeling tools and environments to run this simulation. PCM models contains structure of the software (e.g. components and interfaces), the behavior (communication methods), the required resource environment, the allocation of software components and the usage profile. Service Effect Specification (SEFF) in PCM captures the behaviour of component using different control flow mechanism, internal actions and external call actions (call to services like loop and branching).

2.4 Reverse Engineering

Component-based mixed-technology software systems keep on updating their components and services regularly to achieve the service quality. It is hard to keep architecture models of these systems consistent with the on-going implementations. In such cases, architecture level performance prediction using PCM become challenging. One way to get the PCM performance model is to use reverse engineering (RE) approach.

RE approaches follow static, dynamic or hybrid analysis methods to extract the architecture from the source code and other artefacts. Reconstruction of PCM performance model by SoMoX approach is purely static extraction approach, so the resulting model does not contain any run-time performance parameters such as resource demand [2]. Langhammer introduced RE tools Extract and EJBmoX [14] that extract the behavior of the source code based on hybrid analysis. He combined the static analysis and extracted the model and later enhanced it with dynamic analysis. Spinner [21] extracted PCM performance model based on dynamic analysis. Finally, Mazkatli et al. [16] have included the estimation of parametric dependencies into the hybrid extraction by Langhammer and extended the hybrid extraction approach to extract models incrementally to be applicable in a continuous integration pipeline. However, all these approaches are specific for a certain mapping between source code and architecture model and do not support the general architecture of mixed-technology software extraction.

In an earlier short paper, we sketched the challenges of extracting architecture models from mixed-technology systems and presented the idea to use common concepts between technologies to structure the extraction rules [20]. In this work, we have developed the approach further and present our approach of a rule engine which also considers dynamic analysis.

2.5 Message-Based Communication

Mixed-technology architecture often uses messaging middleware for data and message passing to address non-functional requirements, e.g. reliability and availability [18]. It uses message channel, a component in the middleware, for this asynchronous loosely-coupled communication. Different services and system components binds to the messaging middleware dynamically during run-time. Messaging middleware also controls the data flow in such systems

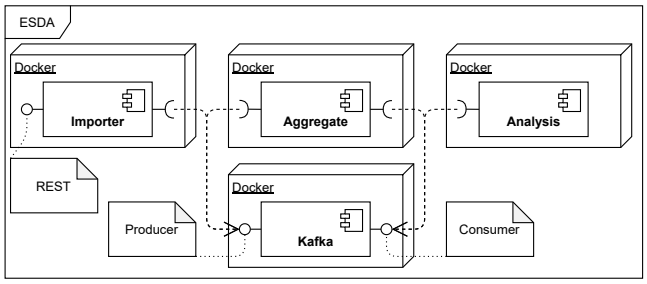

Figure 1: Services and their deployment of the ESDA system.

and has a decisive influence on the performance of the system. In order to complete performance model of the systems with messaging middleware, it is important to capture the temporal behaviour of such communication during run-time. Automatic reconstruction of the performance parameters from system implementation is helpful in this context. To the best of our knowledge there exists no such formal approach which support the automatic reconstruction of architecture-level performance prediction for mixed-technology software systems particularly talking via message-based middleware.

3 LONG-TERM VISION

The long-term vision of the approach we are pursuing is to automatically create and calibrate a performance prediction model as part of the continuous integration / continuous deployment pipeline of modern software development. Once the performance model is automatically created, it can be used to predict performance and answer open design decision questions. These model-based performance tests are intended to identify problems as early and as cost-effectively as possible. In doing so, evaluating performance requirements at the model level should reduce the cost of a complete performance test environment.

Since we want to model and take advantage of the knowledge of the technologies used, we expect that our proposed approach will generate performance prediction models that better fit the actual architecture and performance characteristics. Likewise, by explicitly modeling the established concepts of component-based systems, we expect the user of the performance prediction model to better understand the relationships between a technology with the underlying concepts.

4 RUNNING EXAMPLE

Energy State Data Analysis (ESDA)[1] is an open-source component-based reference and test application for use in performance benchmarks and testing. This small case study emulates a message-based system for analyzing energy state data. ESDA is a distributed microservice application with three distinct services plus a Kafka broker. Each service can be replicated indefinitely and can be deployed on different hardware nodes. The services communicate mainly through Spring's standardized consumer and producer interfaces for Kafka and can be configured through Spring's standardized interfaces.

[1]https://github.com/kit-sdq/esda

The figure 1 shows the services and their deployment in UML notation. The `Importer` service is responsible for receiving and sending measurement data via a REST interface to the `Kafka` broker. The `Aggregate` service is responsible for aggregating measurement data. To do this, it receives multiple measurement data from the Kafka broker, aggregates a specified number of them into a new value, and sends it back to the Kafka broker. In this context, the number specified in this way creates a so-called parametric dependency, which describes the relationship between the input parameters of a component and its performance characteristics. The `Analysis` service emulates a possible analysis of this aggregated measurement data.

The ESDA case study was explicitly implemented with regard to the use of the most common technologies. Thus, it is intended to be a sample of the most common message-based architecture using modern technologies. For example, Kafka is used as the message broker, and the services with message-based communication are built entirely on the Spring framework and deployed in Docker containers. In the further part, we will use this case study to show how we want to use knowledge about used technologies to derive a model of this message-based architecture for performance prediction.

5 APPROACH

The focus of our proposed approach is on the recovery of component-based architectures from projects. For this purpose, an already developed component-based system must be available, which consists of source code and additional artifacts such as Docker files. A pure reconstruction of models to predict the performance of the component-based system using only the source code is often not possible anymore. Therefore, the idea of our proposed approach is to bring in artifacts from different information sources for the reconstruction and then transform them into a model for performance prediction.

5.1 Rule Engine

The idea of our proposed approach is to develop a rule engine for architecture reconstruction with a focus on mixed-technology software. The main contribution of the approach can be divided into two tasks. First, the engine itself, which parses the rules provided later and applies them to generate PCM models. Second, the definition and creation of rule artifacts for component identification.

For example, several frameworks are available for component-based development. Some frameworks support explicitly marking parts of the source code as components like the Java annotation `@Component` in Spring. In Spring, not only components but also their provided interfaces can be precisely identified by annotations like `@RequestMapping`. This would allow one to easily identify all components in a Spring-based framework by searching for the appropriate annotations.

This fact that modern component-based mixed-technology software systems built on top of a given framework or library are likely to use the same standardized mechanisms as annotations is the basis for component detection in our proposed approach. This leads to the idea that such component recognition structures can be formulated and collected as rules. Each rule thus represents a pattern

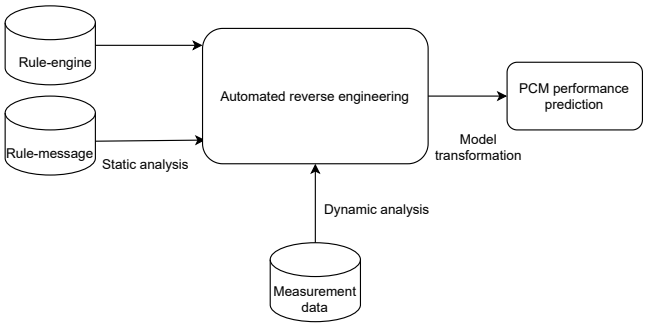

Figure 2: Overview of the proposed approach

for identifying and mapping source code parts to a component, an interface, parameters, or other parts of a model for predicting the performance of CBSE.

Before the actual implementation can begin, a structure for the precise formulation of transformation rules must be defined. For this purpose, different technologies that support this goal have to be compared. In addition, the formulation mechanism for these rules must be simple enough to ensure good usability for users. Additionally, the rules should use fully qualified names to avoid collisions with e.g. annotations with the same name. Since existing transformation languages tend to take structures from common programming languages, a user might face a steep learning curve when trying to formulate their own rules. Therefore, one idea could be to define a DSL (Domain Specific Language) for one of the proposed transformation languages. This would result in using already established transformation ecosystems while providing users with a more readable and thus understandable way of formulating rules. The rules would be designed that they can be reused across project boundaries. This means, for example, that the Spring Components rules for all software projects can identify the components that were implemented with the Spring framework.

5.1.1 Parsing. The artifacts in the software development of a component-based system should first be transformed into appropriate models suitable for further processing as an intermediate step. The approaches, which are used for parsing the information and transfer into the intermediate model, can differ from source to source. For this purpose, we are currently working for enabling Java source code and Docker configuration files to be used as artifacts. Then, the transformed artifacts along with already formulated rules lead to a transformation into model for performance prediction representing the software architecture.

The text-to-model transformation approach differs for the two information sources. For the Java source code, we use an existing Eclipse JDT[2] parser. The information of the abstract syntax tree obtained from this will be transformed into a previously defined EMF model. For the Docker configuration files, a classic text-to-model transformation is to be applied using Xtext[3]. The information of the configuration files is to be transferred with it by an appropriate grammar into a model, whereby Xtext creates the model.

[2]https://www.eclipse.org/jdt
[3]https://www.eclipse.org/Xtext

5.1.2 Asynchronous Message Communication. The current on-going work on rule engine supports only RESTful services with a focus to Spring applications. We will extend the rule engine to extract the mixed-technology architectures based on asynchronous message communication. message-based communication comes in two flavours: point-to-point and publish-subscribe. In case of message-based communication the data management and message routing is decentralized. The service locator component integrates the services and data elements. The interaction method is commonly enabled by the smart end points. The message queues and message-based middleware often perform indirect communication. We aim to capture and model the messaging communication by two ways, as architects may prefer different views on the system depending on their current information needs.

Messaging as a basic component Here the messaging component, such as Kafka, will be represented as a so-called basic component in the architecture model, similarly to other, logical components of the system. Considering ESDA system in which Importer component communicates to Analyse via Kafka would be represented using three components `Importer`, `Analysis`, and `Kafka` in the architecture model where `Importer` is connected to `Kafka` and `Kafka` is connected to `Analysis`. The representation communication via Kafka is presented in Figure 1. We will generate an aggregation function to aggregate the run time information to the messaging component and provide a high-level view of the system. We will enable an additional service to collect the request and associated responses. This will collect the service-to-service communication of particular message-based interactions, deriving an architecture diagram in terms of components and communication relations.

Messaging as middleware The idea here is to consider message-based communication as indirect communication. It is possible to model the asynchronous message exchange via message channels. In case of our running example ESDA, the Importer component communicates to the Aggregator component via Kafka would be represented using only two components `Importer` and `Aggregator` and an explicit model element for a message channel via Kafka in the architecture model. In this case, `Importer` is connected to `Aggregator` and this connector is annotated to use Kafka (using the event-based modelling approach for PCM by Rathfelder et al. [8]) (cf. Fig. 3). Before performance simulation, the model transformation by Rathfelder et al. then inserts adapters and middleware components as shown in Fig. 4 as a so-called model completion. Exchanging messages between components describes a dataflow from one component to another, which is critical for the understanding of the system in total. We plan to tackle this understandability to the reconstructed performance model by modeling the message exchange in the direction of the data flow.

The following example briefly illustrates how we intend to use knowledge of a particular technology to automatically discover message-based communication between services from source code, and thus automatically generate parts for a architecture-based performance prediction model.

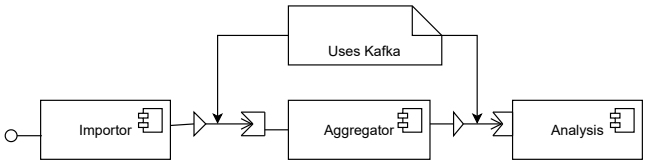

Figure 3: Modelling messaging as a middleware, using syntax defined by Rathfelder et al. [8] for event connectors

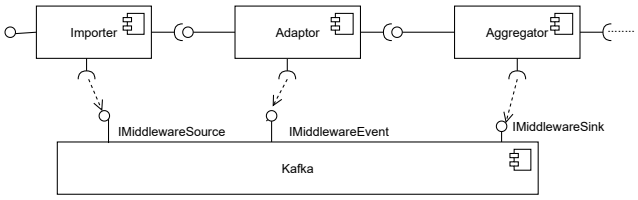

Figure 4: Completed model for simulation

```
@Component
public class EsdaListenerController {
  // ...
  @KafkaListener(topics =
       "esda.aggregate.topic − name", clientIdPrefix = "
       {spring.kafka.consumer.client-id}",
    containerFactory = "kafkaListenerContainerFactory")
  public void listenAsObject(
    final ConsumerRecord<String, Measurement> cr,
    @Payload final Measurement payload) {
    // ...
    template.send(topicName, data);
    // ...
  }
}
```

Listing 1: Snippet of source code for a Kafka receiver.

Listing 1 shows a shortened snippet of source code for a Kafka receiver implemented using the Spring framework. This receiver is used to receive new measurements in the aggregate service from the ESDA case study. As well, shows a snippet for a Kafka sender, which is used to send aggregated measurement values in the same service.

5.2 Dynamic Analysis

In our subsequent work, we want to combine both static analysis and dynamic analysis to extract the architecture.

Static analysis The goal of static analysis is to identify which message channel a component under study communicates with. The result should be an assignment of the sender-receiver relationship. Our idea is to examine the calls of the communication methods in the source code. By the planned annotation of the messaging-API, the signature of the methods and also the parameter types will be known. The analysis will read the concrete variable instance of the parameters and examine the rest of the source code for the value of the variable instance. However, the static analysis may not be sufficient to discriminate between different possible receivers of a message, thus we augment the analysis by dynamic analysis.

Dynamic analysis We plan to base the dynamic analysis on communication methods and their signatures, as well as on knowledge of the messaging-API annotations. However, there will be no parser that extracts the values of the parameters from the source code. So, we will use the logging mechanism that writes the values to a log or will monitor the files at run-time. On these values, the analysis will augment the sender-receiver relationships found by the static analysis. Further, we will instrument the source to refine the SEFF. It will provide us the most exact PCM performance model.

The internal behavior of large message-based systems cannot be determined by static analysis alone. The Kieker Monitoring Frameworks [22] provides additional dynamic analysis capabilities, i.e. monitoring and analyzing the runtime behavior of a software system. The Performance Model eXtractor (PMX) [23] tool automates the extraction of architectural performance models from log files using Kieker as input data format. PMX separates the learning of generic aspects from model creation and is able to extract PCM models purely based on dynamic analysis. PMX provides a solution that integrates established tools for monitoring, log processing, and resource requirement estimation. However, PMX requires existing source code to be instrumented and executed. Here, Kieker provides several instrumentation options for control flow tracking, e.g., intercepting middleware for the Spring framework.

In our proposed approach, we plan to use technology-specific knowledge to set instrumentation for tracking message-based communication at relevant points in the source code. In the same way, we plan to replace the automatic learning of generic aspects with the results of the rule engine, so that the PCM model is calibrated by the dynamic analysis. The extracted architecture via the rule engine will be transferred to the target PCM model to predict the performance of the system. The discussed overall approach can be envisioned through Figure 2.

6 PLANNED EVALUATION

For our evaluation, we will use open-source GitHub projects for which the architectural documents are available. Using our approach, we will extract the architecture from the source code of the selected GitHub projects. We will then enrich the architecture with the performance parameter gathered from run-time data. Later we will compare the final PCM performance architecture extracted by our approach with the architecture provided by the project and consider how accurately components and communication links were extracted. We will use precision and recall metrics to measure the results. Additionally, we will evaluate the accuracy of the performance prediction approach. Both evaluation aspects will quantify the extraction process answering our guiding question "how well our approach will extract the architecture level performance model of mixed-technology software systems."

7 RELATED WORK

In the context of performance model extraction there exists several publications. Walter et al. [23] propose a tool to extract an architectural performance model and performance annotation based on analysing monitoring data. Krogmann [13] extracts parametrized PCM performance model based on hybrid analysis. These extractions particularly do not support mixed-technology architecture and do not include the domain knowledge of various technologies during static analysis.

A number of RE approaches available and can be used for architecture extraction [7]. Often these approaches are limited in the case of mixed-technology architectures particularly communicating via messaging middleware. RE tools like Sotograph [4], Zipkin [9] are available for architecture recovery from source code. However, they only provide the information about component dependencies present in the source code. The recovered architecture largely misses the performance attributes required to model PCM performance model.

A wide range of works are available dealing with performance model with special attention to message-based communication [6, 8]. Happe [8] modelled the message-based communication considering platform specific middleware. Later Rathfelder [17] extended the approach with PCM meta model to explicitly model asynchronous communication. They provide us a good collection of performance related attributes and modelling approaches with PCM for message-based communication. But, we in our work focus to extract these performance attributes from source code. We are trying to generate such PCM model through a RE approach by capturing these performance attributes and SEFF from dynamic analysis which has not been tried yet.

8 CONCLUSION

In this work, we propose the development of a novel approach for recovery of the architectural-level performance model for mixed-technology software systems communicating via messaging middleware. This is achieved by two-step approach, first, static extraction of the architecture based on domain-knowledge of the technologies used (rule engine) in our case, second, refining the extracted architecture with performance attributes measured from run-time (dynamic analysis). The refined architecture obtained as the final result of these two steps provide us with a performance model which conforms to PCM. The generated performance model is then simulated with PCM for performance simulation and prediction.

The major contributions of our approach are: first, automatic extraction of architectural-level performance model for CBSE communicating via message, second, suitable reverse engineering approach with behavioural extraction along with static performance attributes, third, re-useable rule engine to encapsulate the technical knowledge for upcoming mixed-technology software systems. In our future work, we aim to integrate our approach into a continuous integration pipeline to enable our reverse engineering approach.

REFERENCES

[1] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. 2004. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering* 30, 5 (2004), 295–310. Publisher: IEEE.

[2] Steffen Becker, Michael Hauck, Mircea Trifu, Klaus Krogmann, and Jan Kofron. 2010. Reverse engineering component models for quality predictions. In *2010 14th European Conference on Software Maintenance and Reengineering*. IEEE, 194–197.

[3] Steffen Becker, Heiko Koziolek, and Ralf Reussner. 2007. Model-based performance prediction with the palladio component model. In *Proceedings of the 6th international workshop on Software and performance*. 54–65.

[4] Walter Bischofberger, Jan Kühl, and Silvio Löffler. 2004. Sotograph a pragmatic approach to source code architecture conformance checking. In *European Workshop on Software Architecture*. Springer, 1–9.

[5] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. 2011. Automated extraction of architecture-level performance models of distributed component-based systems. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 183–192.

[6] Adnan Faisal. 2017. *A Flexible Framework for Modeling Middleware Completions*. PhD Thesis. Carleton University.

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

[8] Jens Happe, Steffen Becker, Christoph Rathfelder, Holger Friedrich, and Ralf H. Reussner. 2010. Parametric performance completions for model-driven performance prediction. *Performance Evaluation* 67, 8 (2010), 694–716. Publisher: Elsevier.

[9] Stefan Haselböck and Rainer Weinreich. 2017. Decision guidance models for microservice monitoring. In *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 54–61.

[10] George T. Heineman and William T. Councill. 2001. Component-based software engineering. *Putting the pieces together, addison-westley* (2001), 16. Publisher: Springer.

[11] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software* 35, 3 (2018), 24–35. Publisher: IEEE.

[12] Samuel Kounev, Fabian Brosig, and Nikolaus Huber. 2014. *The descartes modeling language*. Universität Würzburg.

[13] Klaus Krogmann. 2012. *Reconstruction of software component architectures and behaviour models using static and dynamic analysis*. Vol. 4. KIT Scientific Publishing.

[14] Michael Langhammer, Arman Shahbazian, Nenad Medvidovic, and Ralf H. Reussner. 2016. Automated extraction of rich software models from limited system information. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 99–108.

[15] Kung-Kiu Lau and Zheng Wang. 2007. Software component models. *IEEE Transactions on software engineering* 33, 10 (2007), 709–724. Publisher: IEEE.

[16] Manar Mazkatli, David Monschein, Johannes Grohmann, and Anne Koziolek. 2020. Incremental Calibration of Architectural Performance Models with Parametric Dependencies. In *IEEE International Conference on Software Architecture (ICSA 2020)*. Salvador, Brazil, 23–34.

[17] Christoph Rathfelder, Benjamin Klatt, Kai Sachs, and Samuel Kounev. 2014. Modeling event-based communication in component-based software architectures for performance predictions. *Software & Systems Modeling* 13, 4 (2014), 1291–1317. Publisher: Springer.

[18] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke. 2015. *Middleware for internet of things: a survey. IEEE Internet Things J. 3, 70/95 (2016)*.

[19] Ralf H. Reussner, Steffen Becker, Jens Happe, Robert Heinrich, and Anne Koziolek. 2016. *Modeling and simulating software architectures: The Palladio approach*. MIT Press.

[20] Yves R. Schneider and Anne Koziolek. 2019. Towards Reverse Engineering for Component-Based Systemswith Domain Knowledge of the Technologies Used. In *Proceedings of the 10th Symposium on Software Performance (SSP) (Softwaretechnik Trends)*. 35–37.

[21] Simon Spinner, Jürgen Walter, and Samuel Kounev. 2016. A reference architecture for online performance model extraction in virtualized environments. In *Companion Publication for ACM/SPEC on International Conference on Performance Engineering*. 57–62.

[22] André Van Hoorn, Jan Waller, and Wilhelm Hasselbring. 2012. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC international conference on performance engineering*. 247–248.

[23] Jürgen Walter, Christian Stier, Heiko Koziolek, and Samuel Kounev. 2017. An expandable extraction framework for architectural performance models. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. 165–170.