Effects of Event-driven Architecture on Modularity Software: A Research Agenda

Luan Lazzari
PPGCA, University of Vale do Rio dos Sinos
São Leopoldo, Rio Grande do Sul, Brazil
luanlazzari@edu.unisinos.br

Kleinner Farias
PPGCA, University of Vale do Rio dos Sinos
São Leopoldo, Rio Grande do Sul, Brazil
kleinnerfarias@unisinos.br

ABSTRACT

Event-driven architecture has been widely adopted in the software industry, emerging as an alternative to modular development to support rapid adaptations of constantly evolving systems. However, little is known about the effects of event-driven architecture on performance, stability, and software monitoring, among others. Consequently, professionals end up adopting it without any empirical evidence about its impact. Even worse, the current literature lacks studies that point to which emerging research directions need to be explored. This article proposes an agenda for future research based on the scarcity of literature in the field of event-oriented architecture. This agenda was derived from a literature review and a case study carried out, as well as from the authors' experience. Five main topics were explored in this work: software performance, empirical studies, architectural stability, challenges for software adoption, and monitoring. Finally, this article seeks to help researchers and professionals by proposing an agenda that serves as a starting point for their research.

KEYWORDS

Event-driven architecture; EDA; agenda; future works; challenges.

1 INTRODUCTION

The development of software systems currently takes place in increasingly unstable business environments, requiring high flexibility to support rapid system adaptations [30]. Typically, complex and volatile business rules, changes in used technologies, pressures for shorter development cycles and continuous delivery of functional modules [34] are some of the ever-present features of contemporary software development projects. For this reason, software development teams seek to use architectural styles, patterns, and software design principles to keep the stability of software system design under development or under maintenance [12]. In addition, development teams seek to promote collaborative development, reduce development and cognitive effort [20, 21], improve effort estimates [6], and minimize the effort invested to integrate and maintain critical software artifacts for the development of complex systems [4, 13, 14]. In this sense, D'Avila [9] also outlines some findings about the effects of contextual information on the reduction of maintenance effort. Even worse, the lack of documentation for the design of software systems has been a factor that makes it even more difficult to maintain current software systems [15, 23]. This absence of UML models in software projects, in part, is motivated by the difficulty of objectively evaluating the generated UML models [16] and keeping such UML models up to date with each other [8] and with the current version of the source code.

Some works [5, 25, 37, 38] point out that event-driven architecture promotes loose coupling — essential for the modularization of application services — but can increase design complexity and system understanding [17]. The software industry has adopted the use of events in architectures. For example, Spotify introduces streaming event delivery to batch to support its applications [1]. Recent studies [25, 27, 38] point out the possible benefits of event-driven architecture. Laigner *et al.* [27] report an empirical study in which the adoption of event-driven architecture improved the maintenance and fault isolation in a system that was refactored after years of maintenance, giving rise to a large and complex source code.

The literature on event-driven architecture advocates that designing applications strongly based on events favors the functionality modularization, as well as facilitating maintenance activities and service evolution of applications [5]. In this sense, designing software adopting event-driven architecture may imply a more systematic way to promote a better modularization of modern software. Moreover, event-driven architecture has been widely adopted in the software industry, emerging as an alternative to modular development to support rapid adaptations of constantly evolving systems. However, little is known about the effects of event-driven architecture on performance, stability, and software monitoring, among others. Consequently, professionals end up adopting it without any empirical evidence about its impact. Even worse, the current literature lacks studies that point to which emerging research directions need to be explored.

This article, therefore, aims to identify the challenges, implications, and future research directions regarding the use of event-driven architecture in the field of software development. For this, we carried out a scoping review in the literature, selecting works related to event-driven architecture for convenience, as there are few public studies on the areas covered, they adopt well-known research guidelines [39]. Moreover, a case study on the software modularity of event-driven architecture was designed and run. Based on these studies, we propose a research agenda that can be exploited to foster future research as well as initiatives in the software development industry. As a result, the article seeks to promote the study and adoption of event-driven architecture. In particular, researchers and professionals can benefit from this work. Researchers can benefit from this research by using it to build their research agendas, direct research efforts, and serve as a guide or starting point for students. Professionals can use this article as support to improve software development practices, drive architectural improvements, and promote software redesign.

This study is structured as follows: Section 2 introduces the main concepts for understanding the event-driven architecture; Section 3 addresses related works, exploring the comparing them

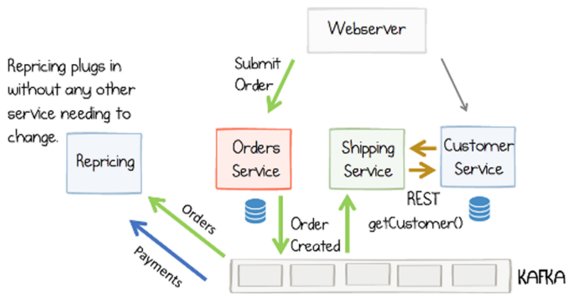

Figure 1: Flow of events in the ordering service (source [37])

with the present one; Section 4 describes the methodology for developing this study; Section 5 brings the proposed agenda; and, finally, Section 6 draws some conclusions and future works.

2 EVENT-DRIVEN ARCHITECTURE

In event-driven architecture, software components publish data without knowing the other components or which can consume and react to the published data, promoting the separation of computation and event publishing from any subsequent processing [17, 25], as illustrated in Fig. 1. Furthermore, the communication between the producer/consumer is asynchronous, and both are independent of each other [11]. Consequently, promoting loose coupling between components — that is the reason event-driven architecture has become predominant in large-scale distributed applications [17].

In addition, three-dimensional visual reconstruction using trains of events with very high temporal resolution, simulation of spiking neural networks and integration of multi agent systems are among other applications that EDA plays an important role. Such applications have some requirements in common, such as responsiveness, event asynchronicity, and heterogeneous data source [35].

The messaging system allows building loosely coupled services, as it moves the raw data to a highly coupled location (producer) and places it in a loosely coupled location (consumer) [37]. Therefore, any operations that need to be performed on this data are not done on the producer, but on each consumer [37]. That is, services can easily be added to the system in plug and play (pluggable) mode, where they connect to event streams and run when their criteria is met [25, 37]. It does not only promote loose coupling, but also manage to store events and data, dispensing with the use of a database, keeping events “close” to the services [37]. In addition, all events are stored in the order they arrived, allowing events to be played back in order. As a result, the performance of event-based applications is also better, ensuring stability and high performance for high data flow [37].

Its composition generally comprises components that detect events, listen for events, process the reaction to an event, and transmit events or messages between components. Event-driven architecture is extremely loosely coupled and highly distributed by design [5]. On the other hand, decoupling between producers and consumers makes controlling the visibility of data more difficult. Although there are few studies addressing the disadvantages or difficulties with EDA, some points prove to be quite challenging.

3 RELATED WORK

This section discusses works that are close to the objectives of our article. The selection of related works was carried out following two steps: (1) search in digital repositories, such as Google Scholar for articles applying the search string “event-driven architecture OR event-driven OR EDA”; and (2) filter of selected articles considering the alignment of such works with the objective of the work and the formulated research questions (Section 4.1). We selected five articles from the literature for convenience, using the criterion of proximity to the topic explored in our research. Such works are analyzed (Section 3.1) and then compared with the proposed work, aiming to identify research opportunities (Section 3.2).

3.1 Analysis of Selected Studies

Overeem *et al.* (2021) [31]. The study analyzes 19 event-sourced systems to understand the reasons for using the event sourcing pattern. Its conceptualization is addressed for better understanding, as well as differentiating from similar architectures such as event-driven, although very similar, they present differences on event concepts. To understand the rationale for adopting event sourcing, it was based on interviews with 25 event sourcing engineers. Through the analyzed systems, three topics associated with event sourcing emerged, the use of Domain-driven Design (DDD) as software design, Command and Query Responsibility Segregation (CQRS) being a related architectural standard and microservice as a style. In each topic the reasons for application are discussed. In addition to the favorable points, five challenges faced by professionals are discussed: event system evolution, the steep learning curve, lack of available technology, rebuilding projections, and data privacy. Finally, from the insights acquired in the analyzed systems and the challenges found, five tactics and solutions were discovered that support professionals in the evolution of event sourcing systems.

Petrov *et al.* (2021) [32]. It analyzes the notions of event processing, event processing methods, area context, and urgent issues of event processing methods. Furthermore, approaches to its solution are proposed, differences in event processing methods are exemplified, and the disadvantages of the methods are highlighted. Emphasizing the following problems: out-of-sequence event processing; occurrence of duplicates; collisions in event processing; fault-tolerant distributed architecture; multithreaded event processing; adaptive load balancing circuits; event processing application monitoring. Various solutions are discussed and tested using the test bench in order to assess the consequences. Some methods can lead to performance degradation.

Schmidt *et al.* (2008) [36]. It conducted a survey of the current state of the art in event-driven architecture, with a focus on event and action processing. Where is described the prerequisites of an entirely new conceptual model to describe the reactivity that is closest to the way people react to events: based on the ability to identify the context during which active behavior is relevant and the situations in which it is necessary. Challenges for event processing are addressed as a way to manage a very valuable knowledge asset - knowing how to react (make decisions) in event-driven situations.

By distinguishing between a non-logical and a logic-based view when dealing with event-triggered reactivity.

Griffin and Pesch (2007) [22]. It presents a survey of service-oriented architecture and web services in telecommunications. Its have gone through several changes and technological evolution, arising from regulations and competition. The article describes the changes in detail and shows that the need to adopt service-oriented architecture (SOA) in telecommunications has become an important item on the agenda of operators. To make this possible, event-driven architecture (EDA) is covered in detail, as SOA and EDA complement each other and are necessary in a real SOA implementation. While SOA provides a request/response message exchange, EDA is capable of long-term asynchronous processing.

Chung-Sheng (2005) [29]. It conducted a study on the evolution of event-driven applications and discusses their potential implications for system and middleware trends. Applications include: telecommunications services, trading system for financial services, logistics and asset management in manufacturing, digital fuel field for oil and gas production, telematics for automotive maintenance, and disease monitoring for healthcare. There is also the analysis of new middleware components and system architectures optimized for event processing and routing.

3.2 Comparative Analysis and Research Opportunities

Comparison Criteria. Seven Comparison Criteria (CC) were defined to identify the similarities and differences between the proposed work and the selected articles. This comparison seeks to help identify research opportunities using objective rather than subjective criteria. The criteria are described below:

- **Agenda research (CC1):** understands studies to address future work or agenda research;
- **Event-driven architecture (CC2):** studies that address concepts or applied the event-driven architecture;
- **Empirical Study (CC3):** studies that performed experimental studies, especially through case studies, experiments, or observations for data collection in the field;
- **Performance analysis (CC4):** studies where performance analyzes were applied;
- **Stability analysis (CC5):** studies where software stability was measured;
- **Challenges to adopting (CC6):** studies that explore the challenges for the adoption of event-driven architecture;
- **Logs (CC7):** studies covering techniques for log analysis in event-driven architecture applications.

Research opportunities. Table 1 presents the comparison of the selected studies, highlighting the similarities and differences between them. To sum up, the analyzed works do not present a purposeful research agenda which can serve as a starting point for future works. Therefore, this work seeks to fill this gap in the current literature.

4 METHODOLOGY

This section defines the methodology used to retrieve the state-of-the-art literature about research agenda regarding the subject of event-driven architecture in software engineering. From this selected literature, we derived the further challenges and implications of applying the event-driven architecture in industry. Section 4.1 presents the objective and research questions of this work. Section 4.2 presents the strategy to select the potential studies. Section 4.3 outlines the process of study selection. Section 4.4 briefly describes the exploratory case study run to identify the benefits of using event-driven architecture in terms of modularity.

4.1 Objective and Research Questions

This study has two main objectives: (1) to grasp a research agenda in relation to the event-driven architecture in software engineering; and (2) to list which implications that the software industry needs to overcome. For this, this study comprises in one research question: How to propose a research agenda containing challenges, implications, and future directions regarding event-driven architecture?

To answer this question, a scoping review was conducted to select systematically the related literature.

4.2 Search Strategy

This section presents the strategy used to search studies in the current literature. The strategy consists of building a search string and defining a main search engine to conduct the search.

Search string. The search string for related works was constructed using the main terms of event-driven architecture. The search string used was as follows:

```
“event-driven architecture OR event-driven OR EDA”
```

In addition to works related to event-driven architecture, works referring to other fields of research are also returned. To address the topics on the agenda, specific research was carried out on each topic that generated other search strings in addition to the one mentioned above, aiming at the most relevant works regardless of the technologies involved.

Search engine. In this work, Google Scholar was the search engine used in the selection of works. This engine was selected because it encompasses works published in the main magazines and congresses related to computer science, which contain works related to event-driven architecture.

4.3 Study Selection

The criterion for selecting the works, in short, was for convenience. Because, the area that studies event-driven architecture still has few works that address the themes explored in this study. On top of that, most of the jobs returned by the search engine were related to other fields of research. Therefore, it becomes impossible to apply the selection of works as some methodologies apply based on this return. Thus, the selection criterion was the title of the article and its abstract, enough points to understand if the article addresses the research objectives.

4.4 Case Study

As previously mentioned in Section 1, event-driven architecture has been widely adopted in the software industry, thus emerging as

Related Work	Comparison Criteria						
	CC1	CC2	CC3	CC4	CC5	CC6	CC7
Proposed work	●	●	●	●	●	●	●
Overeem *et al.* (2021) [31]	○	○	○	●	○	●	○
Petrov *et al.* (2021) [32]	○	●	●	●	○	●	○
Schmidt *et al.* (2008) [36]	○	○	○	○	○	●	○
Griffin and Pesch (2007) [22]	●	●	○	○	○	●	○
Chung-Sheng (2005) [29]	○	●	○	●	○	○	○

● Attends ○ Does not attend

Table 1: Comparative analysis of selected relate works

an alternative to the development of enterprise applications based on REST architectural style, for example. We realize that little was known about the effects of event-driven architecture on software modularization while enterprise applications evolve. Consequently, practitioners ended up adopting it without any empirical evidence about its impacts on essential indicators, including separation of concerns, coupling, cohesion, complexity and size [28]. We designed and run an exploratory case study to compare event-driven architecture and REST style in terms of modularity. A real-world application was developed using an event-driven architecture and REST through five evolution scenarios. In each scenario, a feature was added. The generated versions were compared using ten metrics. The initial results suggest that the event-driven architecture improved the separation of concerns, but was outperformed considering the metrics of coupling, cohesion, complexity and size. The findings are encouraging and can be seen as a first step in a more ambitious agenda to empirically evaluate the benefits of event-driven architecture against the REST style.

This exploratory case study explored in practice the possible benefits of using event-driven architecture, which served to build the proposed research agenda.

5 PROPOSED RESEARCH AGENDA

This section presents the proposed research agenda, which was structured through five perspectives, including performance analysis, conducting empirical studies (Section 5.1), stability analysis (Section 5.3), challenges to adopting the technology (Section 5.4), and logging (Section 5.5).

5.1 Performance analysis

The main characteristic observed in a distributed system is your **process capacity**. The performance of a software indicates the degree to which the system fulfills its tasks, measured by the response time and the efficiency with which it achieves it [26]. While response time is the time it takes to respond to a request. Sites are often vulnerable to high traffic, usually on special dates. However, this is often associated with system scalability. Thus, software performance and scalability are essential to avoid system downtime due to overload, something common in today's systems. Certain characteristics can affect these features, e.g., message size, device type, and the complexity of the smart environment in terms of the number of producers and consumers [35].

For that, it is important to explore how the event-driven system should be configured to extract its **scalability** to the maximum [25]. Scalability refers to the software's ability to deliver a continuous response time and throughput as the demand for the service it provides increases and resources are added [25, 26]. It can be vertical or horizontal. Cloud environments allow the infrastructure team to easily configure as many machines as they need to meet demand. In addition to allowing you to optimize operating costs, once the environment is well configured and optimized. The literature [37] explores the capabilities in terms of configuration of event-driven platforms, such as Kafka. As the design and implementation of distributed systems is a challenge, ensuring that the software achieves the analysis objectives is crucial. Therefore, comparative studies on different platforms have become opportunities to explore their capabilities, difficulties and best approach. A key challenge is an initial assessment of performance-related factors, where potential bottlenecks in modeling, design, and implementation can be identified. This can reduce the cost of making any changes [24]. Along these lines, performance comparisons between event-driven architecture and traditional architectures such as REST are also interesting.

The event-driven architecture stands out for its high event processing capacity, handling continuous and **high-volume** data streams [25, 33]. In addition to being chosen for cases that demand scalability, it can be scalable in streaming events, scalable in the volume of historical data, scalable in the number of sources and data collectors, scalable in the number of processing elements, and scalable in terms of physical infrastructure [18, 25]. Moreover, it facilitates the interaction of heterogeneous devices in intelligent environments, where each one operates on its own operating system, communication protocol, a form of interaction, among others [35].

5.2 Empirical studies

Due to the differences pointed out between event-driven architecture and traditional architectures, naturally there must be **difficulties on EDA**. While in traditional software architectures, functions are nested, and it is easy to locate which functions call a given function. In event-driven architecture, events trigger services by each service criterion. Therefore, knowing which functions participate in a given flow becomes more challenging. Therefore, it is essential to know what can happen and how to deal with these difficulties, before the problem happens. The most dangerous time to deal with this is during a crisis, as taking actions without knowledge can make it worse.

Therefore, studies that address the **drawbacks and benefits** of event-driven architecture are essential for adoption. In Laigner *et al.* [27], a monolithic application was replaced, motivated by the difficulty in the maintenance process. The legacy BDS application was replaced by a current Big Data application, some benefits were perceived such as ease of maintenance and fault isolation. However, the complex data flow generated by the amount of microservices, as well as the myriad of technologies, have drawbacks.

The literature presents several case studies involving event-driven architecture. As a Big Data system (BDS) application for oil industry traffic monitoring [27], Euphoria is a new software in event-driven architecture aimed at intelligent environments [35], Digital Twin real-time data stream processing system [2], e-archive document management system [11]. In general, event-driven architecture is used to meet imposed needs, such as modularity, scalability, and asynchronicity for producing, processing, and transmitting messages and events [11, 35].

5.3 Stability analysis

Less change propagation in **maintenance** is desirable for any system. Since software lifecycle costs, it is estimated that between 40% and 67% are related to software maintenance [40]. Therefore, software maintenance can be seen as one of the main attributes of software quality. A simple change in Service A can interrupt a Service B, simply because a function call has some parameter added or removed. Generating some unavailability in these services. In event-oriented architecture, the weak connection between services makes it more difficult to track this, since one event triggers another, when it meets the conditions. Therefore, changes to one service can propagate to others, so documentation about these relationships between services can be helpful.

Once the maintenance represent a high cost in a software lifecycle, stable software is aimed at. For this, studies that explore the **stability** promoted by the event-driven architecture explain benefits in terms of lower costs. The first attribute affected when making a software modification is stability. If the software stability is low, the impact of any modification will be high. Therefore, maintenance will cost more and reliability can also suffer due to the introduction of possible new errors [40]. The stability of a module can be defined as the resistance of a change in one module to affect other modules [40]. Plus, there's logical and performance stability. Where logic measures the impact of a change in one module propagating to another, in logical terms. The performance measures the impact of changes considering the performance [40]. The literature points out that the highest costs involving software are related to the lack of average software measures. Such measures can be attributed to software quality, which is quite generic [40]. Studies in this area have contributed to define some software quality attributes such as correctness, flexibility, portability, efficiency, reliability, integrity, testability, and maintenance. Therefore, stability can be considered a very relevant metric for any software. Because, as studies show, it is directly linked to software maintenance. Comparisons between the stability promoted by event-driven architecture and traditional architectures will be able to show which tends to be less expensive.

This shows that the **software evolution** based on event-driven architecture has considerable challenges to explore. Current literature [5, 37] points out that event-driven architecture promotes loose coupling — essential for the modularization of software services — but can increase design complexity and system understanding [17]. Such modularization aims to isolate the software modules so that changes that occur in one module do not affect the others, so the software would present a better stability. This modularity can be useful in scenarios where services have been added/removed in system software. Since modularity prevents maintenance propagation. This can be measured through metrics collected between versions of a software system's evolution.

5.4 Challenges to adopting

Systems that want to take advantage of the benefits of event-driven architecture will have some challenges. It would be interesting to **trace the types of systems** that benefit from event-driven architecture. A good architecture helps the system meet key requirements in areas such as performance, reliability, portability, scalability, and interoperability. Software architecture plays a key role as a bridge between requirements and implementation [19]. By providing an abstract description of a system, the architecture exposes certain properties while hiding others. Ideally, this representation provides an intellectually traceable guide to the entire system, which allows designers to reason about the system's ability to satisfy certain requirements and to suggest a design for the construction and composition of the system [19]. Sometimes when choosing the best architecture for a system, mistakes can be made when choosing the least suitable one, either because it is a trend or lack of knowledge about its strengths and weaknesses. In Overeem *et al.* (2021) [31] the cause for the chosen event source for the systems under study was explored. Some respondents responded that it was because of the architectural trend and/or curiosity.

Another challenge is its **learning curve,** as event-driven architecture has considerable differences compared to traditional architectures. Building or rewriting systems will require teams composed of architects and a developer capable of handling event-driven architecture are needed to develop a system. However, most IT professionals deal with traditional architecture, and moving to event-driven architecture can be expensive. Garlan [19] points out that software architecture can be useful in at least six aspects of the system: (i) *understanding*: simplifies our ability to understand systems, by the abstraction presented in which the high-level design of a system can be easily understood; (ii) *reuse*: it is divided into several levels, such as reuse of components, as well as structures in which they can be integrated; (iii) *construction*: provides a diagram indicating the main components and dependencies between them; (iv) *evolution*: expose the dimensions along which the system is expected to evolve, as well as assist in maintenance, exposing the unfolding of changes and, thus, more accurately estimating the costs involved; (v) *analysis*: provides opportunities for analysis, including consistency checking, compliance with constraints imposed by an architectural style, compliance with quality attributes, domain analysis, and dependency for architectures built in specific styles; (vi) *management*: successful projects see software architecture as an important milestone in the development process, as it makes requirements, implementation strategies and potential risks much clearer.

5.5 Logs

Capture logs from streams is one of the challenges that could be researched and deepened. Despite efforts to deliver working software, the size and complexity of the software, combined with the real time and budget for development, make it increasingly difficult to deliver fully working software [41]. Software systems inevitably have flaws, such as bugs triggered by some combination of errors, environmental issues, or administrative errors. However, it is not always easy to identify such flaws, which can take time. They are quite worrisome for two reasons: generally they must be resolved quickly, as it is in production, so some part of the software must be inaccessible; second, the difficulty in analyzing the failure, commonly due to lack of data, making reproduction and assertiveness in the treatment impossible. A tool widely used by developers is the System Logs, they make it possible to track and record the behavior at run-time of the software. They are usually used for monitoring, fault diagnosis, performance analysis, test analysis, security and legal compliance and business analysis [7].

Logging consists of two phases: (i) instrumental logging and (ii) log management. Instrumental logging consists of code inserted by developers to record information at run-time. Log management is concerned with analyzing the collected logs to generate relevant information about the behavior at run-time [7]. Therefore, extracting information from logs in log management depends on the quality of the log code produced in the previous phase. The low quality of the log can imply in the diagnosis of problems, high effort in maintenance, low performance, or even software crashes [7]. Therefore, it is important to record the system logs, mainly to identify failures, which can make part of the system unavailable, as well as their analysis to extract relevant data about the system's functioning. To mitigate possible failures, improve performance and even help with maintenance. Tools commonly applied in other technologies can be used to capture logs. However, high event traffic in an event-driven system will likely generate a lot of data, so feasibility studies or better techniques are essential.

As a means of protection to downtime, the service contracting party has Service Level Agreements (SLAs) which are a service contract in which the service level between the customer and supplier is defined. SaaS providers are responsible for ensuring that applications are available 24 hours a day, 7 days a week (24/7), ideally with no downtime. Downtime tolerance in SaaS apps can be less than traditional web apps. The effects of downtime on SaaS applications match the effects of downtime on large-scale e-commerce applications [3]. This can lead to significant financial losses. For that, availability measures are used to define uptime and downtime, such as response time for requests, as well as possible penalties in case of downtime under the provider.

In addition, **distributed log composition** is also worth studying. As several differences can be observed in relation to traditional architectures, mainly in the execution, which occurs individually in each service as opposed to a stack trace as in traditional architectures. Events are triggered by others, there is no traditional stack, therefore, how to identify that a service is not being triggered. Tools like Automated Log Abstraction Techniques (ALAT) can help. However, there is a gap between academia and industry as engineers do not know the best ALAT, due to lack of time and resources to search the literature, and due to lack of studies that describe the best ALAT [10]. Therefore, logs contain abundant data that can help engineers understand the runtime properties of a system. However, large and complex systems produce abundant data to analyze. For this, ALAT can help reduce the data to be processed through its record abstraction algorithms [10].

6 CONCLUSION

Event-driven architecture has the potential architecture for the development of distributed systems, with promising gains in modularization, scalability and concurrency. Some studies show that it has been shown to be effective for various applications due to its characteristics. However, it has few empirical studies about performance, stability, log and challenges to adopting. Studies on these topics can accelerate the adoption by the industry, who has not certainly about the consequences. Therefore, the objective of this work was to provide future directions to researches and practitioners about the use of event-driven architecture on systems.

Overall, the agenda shows that event-driven architecture has many questions to be studied and presented to better understand the consequences of being adopted in software engineering. The challenges also reinforce when the architecture can represent a considerable part of the costs in the software lifecycle. Therefore, if event-driven architecture has the potential to decouple services, it could facilitate the maintenance process, consequently lowering the cost. We hope that this work will serve as a starting point for future research.

7 ACKNOWLEDGMENT

This work was partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant 313285/2018-7.

REFERENCES

[1] [n.d.]. Changing the Wheels on a Moving Bus — Spotify's Event Delivery Migration. https://engineering.atspotify.com/2021/10/20/changing-the-wheels-on-a-moving-bus-spotify-event-delivery-migration/. Accessed: 2021-10-26.

[2] A. B. A. Alaasam, G. Radchenko, and A. Tchernykh. 2019. Stateful Stream Processing for Digital Twins: Microservice-Based Kafka Stream DSL. (2019), 0804–0809.

[3] Sean Banerjee, Hema Srikanth, and Bojan Cukic. 2010. Log-Based Reliability Analysis of Software as a Service (SaaS). In *2010 IEEE 21st International Symposium on Software Reliability Engineering*. 239–248. https://doi.org/10.1109/ISSRE.2010.46

[4] Vinicius Bischoff, Kleinner Farias, Lucian José Gonçales, and Jorge Luis Victória Barbosa. 2019. Integration of feature models: A systematic mapping study. *Information and Software Technology* 105 (2019), 209–225.

[5] Hui Cao, Xing Yang, and Raoyi Deng. 2021. Ontology-Based Holonic Event-Driven Architecture for Autonomous Networked Manufacturing Systems. *IEEE Transactions on Automation Science and Engineering* 18, 1 (2021), 205–215. https://doi.org/10.1109/TASE.2020.3025784

[6] Carlos Eduardo Carbonera, Kleinner Farias, and Vinicius Bischoff. 2020. Software development effort estimation: a systematic mapping study. *IET Software* 14, 4 (2020), 328–344.

[7] Boyuan Chen and Zhen Ming (Jack) Jiang. 2021. A Survey of Software Log Instrumentation. *ACM Comput. Surv.* 54, 4, Article 90 (May 2021), 34 pages. https://doi.org/10.1145/3448976

[8] McLyndon S de L. Xavier, Kleinner Farias, Jorge Barbosa, Lucian Gonçales, and Vinicius Bishoff. 2019. UMLCollab: A Hybrid Approach for Collaborative

Modeling of UML Models. In *Proceedings of the XV Brazilian Symposium on Information Systems*. 1–8.

[9] Leandro Ferreira D'Avila, Kleinner Farias, and Jorge Luis Victória Barbosa. 2020. Effects of contextual information on maintenance effort: A controlled experiment. *Journal of Systems and Software* 159 (2020), 110443.

[10] Diana El-Masri, Fábio Petrillo, Yann-Gaël Guéhéneuc, Abdelwahab Hamou-Lhadj, and Anas Bouziane. 2020. A systematic literature review on automated log abstraction techniques. *Inf. Softw. Technol.* 122 (2020), 106276.

[11] H. Falatiuk, M. Shirokopetleva, and Z. Dudar. 2019. Investigation of Architecture and Technology Stack for e-Archive System. In *IEEE Inter. Scientific-Practical Conference Problems of Infocommunications, Science and Technology*. 229–235.

[12] Kleinner Farias, Alessandro Garcia, and Carlos Lucena. 2014. Effects of stability on model composition effort: an exploratory study. *Software & Systems Modeling* 13, 4 (2014), 1473–1494.

[13] Kleinner Farias, Alessandro Garcia, Jon Whittle, Christina von Flach Garcia Chavez, and Carlos Lucena. 2015. Evaluating the effort of composing design models: a controlled experiment. *Software & Systems Modeling* 14, 4 (2015), 1349–1365.

[14] Kleinner Farias, Alessandro Garcia, Jon Whittle, and Carlos Lucena. 2013. Analyzing the effort of composing design models of large-scale software in industrial case studies. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 639–655.

[15] Kleinner Farias, Lucian Gonçales, Vinicius Bischoff, Bruno Carreiro da Silva, Everton T Guimarães, and Jacob Nogle. 2018. On the UML use in the Brazilian industry: A state of the practice survey (S).. In *SEKE*. 372–371.

[16] Kleinner Farias and Bruno C da Silva. 2020. What's the grade of your diagram? towards a streamlined approach for grading UML diagrams. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–2.

[17] L. Fiege, G. Mühl, and F. Freiling. 2002. Modular event-based systems. *Knowl. Eng. Rev.* 17 (2002), 359–388.

[18] Fabiana Fournier, Alexander Kofman, Inna Skarbovsky, and Anastasios Skarlatidis. 2015. Extending Event-Driven Architecture for Proactive Systems. *CEUR Workshop Proceedings* 1330.

[19] David Garlan. 2000. Software Architecture: A Roadmap. *Proc. of the 22nd International Conference on Software Engineering, Future of Software Engineering Track* (02 2000). https://doi.org/10.1145/336512.336537

[20] Lucian Gonçales, Kleinner Farias, Bruno da Silva, and Jonathan Fessler. 2019. Measuring the cognitive load of software developers: A systematic mapping study. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 42–52.

[21] Lucian Gonçales, Kleinner Farias, and Bruno C da Silva. 2021. Measuring the cognitive load of software developers: An extended Systematic Mapping Study. *Information and Software Technology* (2021), 106563.

[22] D. Griffin and D. Pesch. 2007. A Survey on Web Services in Telecommunications. *IEEE Communications Magazine* 45, 7 (2007), 28–35. https://doi.org/10.1109/MCOM.2007.382657

[23] Ed Júnior, Kleinner Farias, and Bruno Silva. 2021. A Survey on the Use of UML in the Brazilian Industry. In *Brazilian Symposium on Software Engineering*. 275–284.

[24] Razib Khan and Poul Heegaard. 2015. Software Performance Evaluation Utilizing UML Specification and SRN Model and Their Formal Representation. *Journal of Software* 10 (05 2015), 499–523. https://doi.org/10.17706/jsw.10.5.499-523

[25] Sabrine Khriji, Yahia Benbelgacem, Rym Chéour, Dhouha El Houssaini, and Olfa Kanoun. 2022. Design and implementation of a cloud-based event-driven architecture for real-time data processing in wireless sensor networks. *The Journal of Supercomputing* 78, 3 (2022), 3374–3401.

[26] Samuel Kounev. 2008. *Software Performance Evaluation*. American Cancer Society, 1–10. https://doi.org/10.1002/9780470050118.ecse390 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470050118.ecse390

[27] Rodrigo Laigner, Marcos Kalinowski, Pedro Diniz, Leonardo Barros, Carlos Cassino, Melissa Lemos, Darlan Arruda, Sérgio Lifschitz, and Yongluan Zhou. 2020. From a Monolithic Big Data System to a Microservices Event-Driven Architecture. In *46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 213–220.

[28] Luan Lazzari and Kleinner Farias. 2021. Event-driven Architecture and REST: An Exploratory Study on Moadularity. *arXiv preprint arXiv, http://arxiv.org/abs/2110.14699* (2021).

[29] Chung-Sheng Li. 2005. Real-time event driven architecture for activity monitoring and early warning. In *Conference, Emerging Information Technology 2005*. 4 pp.–. https://doi.org/10.1109/EITC.2005.1544382

[30] Anderson Oliveira, Vinicius Bischoff, Lucian José Gonçales, Kleinner Farias, and Matheus Segalotto. 2018. BRCode: An interpretive model-driven engineering approach for enterprise applications. *Computers in Industry* 96 (2018), 86–97.

[31] Michiel Overeem, Marten Spoor, Slinger Jansen, and Sjaak Brinkkemper. 2021. An empirical characterization of event sourced systems and their schema evolution — Lessons from industry. *Journal of Systems and Software* 178 (Aug 2021), 110970. https://doi.org/10.1016/j.jss.2021.110970

[32] Valery Petrov, Anna Gennadinik, Elena Avksentieva, and Konstantin Bryukhanov. 2021. CURRENT ISSUES AND METHODS OF EVENT PROCESSING IN SYSTEMS WITH EVENT-DRIVEN ARCHITECTURE. *Journal of Theoretical and Applied Information Technology* 99 (05 2021), 1943–1954.

[33] Amir Rahmani, Zahra Babaei, and Alireza Souri. 2021. Event-driven IoT architecture for data analysis of reliable healthcare application using complex event processing. *Cluster Computing* 24 (06 2021), 1–14. https://doi.org/10.1007/s10586-020-03189-w

[34] Maluane Rubert and Kleinner Farias. 2021. On the Effects of Continuous Delivery on Code Quality: A Case Study in Industry. *Computer Standards & Interfaces* (2021), 103588.

[35] Ovidiu-Andrei Schipor, Radu-Daniel Vatavu, and Jean Vanderdonckt. 2019. Euphoria: A Scalable, event-driven architecture for designing interactions across heterogeneous devices in smart environments. *Information and Software Technology* 109 (2019), 43–59.

[36] Kay-Uwe Schmidt, Darko Anicic, and Roland Stühmer. 2008. Event-driven Reactivity: A Survey and Requirements Analysis. In *In 3rd International Workshop on Semantic Business Process Management*. 72–86.

[37] Ben Stopford. 2018. *Designing Event-Driven Systems*. O'Reilly Media, Incorporated.

[38] Nico Surantha, Oei K. Utomo, Earlicha M. Lionel, Isabella D. Gozali, and Sani M. Isa. 2022. Intelligent Sleep Monitoring System Based on Microservices and Event-Driven Architecture. *IEEE Access* 10 (2022), 42069–42080. https://doi.org/10.1109/ACCESS.2022.3167637

[39] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[40] S.S. Yau and J.S. Collofello. 1980. Some Stability Measures for Software Maintenance. *IEEE Transactions on Software Engineering* SE-6, 6 (1980), 545–552. https://doi.org/10.1109/TSE.1980.234503

[41] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, and Stefan Savage. 2011. Improving Software Diagnosability via Log Enhancement. *SIGARCH Comput. Archit. News* 39, 1 (March 2011), 3–14. https://doi.org/10.1145/1961295.1950369