Breaking Down Barriers: Building Sustainable Microservices Architectures with Model-Driven Engineering

Gabriel Morais
gabrielglauber.morais@uqar.ca
Université du Québec à Rimouski (UQAR)
Lévis, QC, Canada

Mehdi Adda
mehdi_adda@uqar.ca
Université du Québec à Rimouski (UQAR)
Lévis, QC, Canada

Dominik Bork
dominik.bork@tuwien.ac.at
Business Informatics Group, TU Wien
Vienna, Austria
Université du Québec à Rimouski (UQAR)
Lévis, QC, Canada

ABSTRACT

Despite their promise of circularity and optimized resource consumption, the concrete achievement of sustainability through Microservices Architecture (MSA) faces challenges. Numerous intricate factors can negatively influence MSAs' design and implementation, compromising their economic and environmental effectiveness. We advocate for adopting standard and shared modeling practices to address these challenges. In this paper, we initiate an open discussion on the root causes of these challenges, relating them to the foundational microservices tenets of independence and autonomy. We also propose directions for researchers and practitioners to expand theoretical and practical knowledge of achieving sustainable microservices architectures through model-driven engineering (MDE).

CCS CONCEPTS

• **Software and its engineering** → **Reusability**; **Model-driven software engineering**; **Abstraction, modeling and modularity**; • **Social and professional topics** → **Sustainability**; • **Computer systems organization** → **Cloud computing**.

KEYWORDS

Software Sustainability, Microservices Architecture, Model-driven Engineering

ACM Reference Format:
Gabriel Morais, Mehdi Adda, and Dominik Bork. 2024. Breaking Down Barriers: Building Sustainable Microservices Architectures with Model-Driven Engineering. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3652620.3687799

MODELS Companion '24, September 22–27, 2024, Linz, Austria

ACM ISBN 979-8-4007-0622-6/24/09
https://doi.org/10.1145/3652620.3687799

1 INTRODUCTION

Sustainability in software engineering has often been associated with the energy consumption of hardware components and software longevity [18], which organizations have considered from an economic perspective. This vision has evolved beyond economic aspects and currently includes social, environmental, and technical ones [18], closely relating it to the circular economy [9]. Indeed, *sustainability* aims to create software that "meets the needs of the present without compromising the ability of future generations to meet their own [18]." It is also the "capacity to preserve a system function over an extended period and to be cost-effectively maintained and evolved [31]."

Sustainability and *circular economy* share the fundamental goal of minimizing resource waste and promoting reuse [9]. In this context, circularity focuses on the value retention of artifacts and their production processes, avoiding waste by reusing, renewing, and regenerating them instead of creating new ones [9, 18].

MSA [24] handles system complexity through modularity and independence. Its primary unit is the *microservice*, a small and autonomous service responsible for specific functionality, running in its own process and communicating through lightweight mechanisms [10]. An MSA-based system is a network of fine-grained distributed services, often deployed on a cloud computing infrastructure, that cooperate to achieve complex functionalities [10, 36].

Microservices have characteristics that relate them to system sustainability. First, their fine-grained nature and technological independence [10] tackle the intensive resource use observed in overweight software [19]. Migrating overweight monolithic legacy applications to fine-grained microservices allows integration of just the right amount of functionalities, implementation using the adapted technology, and granular execution on on-demand infrastructures [10, 24]. Second, microservice modularity fosters reuse, which encompasses the consumption of existing instances and the various arrangements in which a microservice can participate, contributing to achieving circularity while reducing the risk of defects and leveraging established expertise for resource-saving and reliable software development. Besides, reuse enactment is simplified by consuming microservices permissionless, in the way that an agreement between consumers and providers is not mandatory [14]. Third, microservices development typically relies on DevOps practices that promote extended process automation, making microservices construction and deployment replicable while allowing DevOps pipelines to be reused [36]. These automatic processes enable the creation of microservices instances on the fly,

supporting on-demand scale-up during operation. Consequently, microservices' circularity extends to replicable development and operation processes, complying with the circular systems engineering bipartite principle introduced in [9].

Finally, virtualization and cloud computing infrastructures allow microservices granular execution with on-demand localized scale-ups to face increased consumption. Similarly, they enable microservices to handle failures, preserving end-users from outages [10]. This ensures systems stability and user trustworthiness, impacting social, economic, and resource usage sustainability tenets [18].

While these microservices tenets contribute to improving software scalability and maintainability and shortening the development cycle, they also introduce a complexity that can hinder sustainability. Microservices' rigorous attention to team independence, reinforced ownership, and slack governance may be barriers to cross-team coordination and effective reuse. In the pursuit of autonomy and decoupling, microservices duplicates may be created, and without proper alignment and governance, this can lead to unnecessary infrastructure scaling and resource overutilization. Thus, the very qualities that make this architecture reusable and reliable can also be its limitation in achieving sustainability. We identified four challenges related to these tenets that model-driven approaches could help solve: 1. lack of standards, 2. local-driven design, 3. lack of coordination, and 4. opportunistic reuse.

This paper explores these challenges' impact on developing sustainable MSA-based systems, highlighting the inherent limitations of current MSA design, modeling, and reuse practices (Section 2). Based on this understanding, we outline perspectives, focusing on MDE, to explore and solve the above challenges (Section 3). Section 4 briefly discusses challenges related to MDE-based MSA. Finally, we outline our future plans and briefly conclude with final remarks (Section 5).

2 CHALLENGES TO MSA SUSTAINABILITY

This section discusses the challenges of accommodating MSA independence, design, permissionless governance, and reuse practices to achieve sustainability. Independence is omnipresent in MSA, from team organization to technological choice for implementation, materializing the "share-nothing" philosophy [24]. This microservice tenet fosters the design of uncoupled applications that can be developed and operated independently, reinforcing team ownership and autonomy. Independence enables effective microservice management and speeds up development cycles [10]. However, various challenges emerge when designing sustainable microservice architectures compliant with the independence tenet [34].

2.1 Lack of Standards

Independence impacts the adoption of shared standards. Indeed, teams adopt modeling practices and documentation techniques that fit their needs. MSA primarily uses informal modeling methods, such as sketches, drawings, semi-formal diagrams (e.g., UML), and "as-code" specifications (e.g., OpenAPI) [21, 36]. Modeling MSA systems using these approaches does not address common modeling challenges like analyzing and exploring multiple viewpoints and modeling in different granularity levels [23]. Therefore, the question of how to formalize, manage, and explore knowledge about these systems arises. Besides, information is fragmented throughout teams, which can hinder the development of 360-degree systems views and the fostering of inter-team communication [1, 23]. Thus, there is a need for conceptual models to support engineers in the early stages of microservices architecture development, allowing them to rely on modeling standards at an abstraction level that enhances explicit microservices interconnections [21] and common understanding [23]. Indeed, no widely accepted and applied formalism to describe MSA has emerged; some works have been proposed [13, 23, 27], but lack evidence of their adoption in practice.

To achieve reuse, comprehensive descriptions of artifacts and mechanisms that match reuse opportunities and reusable parts are mandatory. Reuse retrieval and selection are generally based on describing elements and defining criteria to compare them, leading to specifying a *standard formalism*, which must be widely accepted and used to be effective [17]. These descriptions are then stored in repositories accessible to potential users that rely on identification mechanisms, such as keyword search and similarity-based discovery, to select candidates to reuse [6]. Therefore, a greater understanding of the connections between functionalities and the microservices implementing them, and microservices arrangements are paramount [1]. Such understanding calls for formalized and shared abstraction approaches, which must be able to support information exploration and retrieval [23]. MSA lacks both aspects.

The lack of standards hinders an end-to-end knowledge of MSA-based systems [23], which is necessary to support sustainability [9]. This results in barriers to microservices identification, variability management, and, subsequently, reuse and interoperability [1].

2.2 Local-driven Design

A local view of microservices supports teams' agility, as choices are made based on local needs and knowledge, with limited alignment to external dependencies [14, 37]. Therefore, microservices design is local and bound to a scoped functionality and a team. In this context, microservices are designed in a closed world, leading to architectural weaknesses. Indeed, deficient alignment caused by a limited view results in unanticipated usage levels, which, coupled with automatic scalability mechanisms, lead to resource overconsumption and systems outages [16, 36]. Besides, a local design approach fosters duplication due to ignorance of similar microservices elsewhere in the organization, harming their effective circularity [23]. Tackling duplication implies handling variability [12], which is complex in MSA, as it encompasses technical and organizational concerns in addition to functional ones [1]. Indeed, variability is closely related to the microservice granularity [35], which depends on the bounded context and operational considerations. Thus, MSA granularity is tightly related to teams' concerns and choices.

There are no clear guidelines to help developers define and assess the granularity of microservices [35]. Thus, finding approaches to objectively define microservices granularity based on trade-offs between modularity, enhanced by fine-grained microservices, and operational constraints seems paramount to avoid overweight microservices. However, local-driven design may be a barrier to achieving optimal granularity because some external dependencies, i.e., not owned by the team, may introduce constraints that are ignored during design. Such a situation underscores the need for enforced

alignment, an organization-wide perspective on microservices, integrated and coordinated design, and close team collaboration [34].

2.3 Lack of Coordination

Microservices are developed in organizational silos, with teams coordinating, when necessary, only with immediate dependencies [36, 37]. This way of working impacts microservices' capacity to be reused [34], and represents a threat to runtime capacity design, as the team's knowledge may be restricted to microservices they own and use [14]. Ignoring who consumes their microservices and their impacts on runtime resource usage may lead to inappropriate design and technological choices. Moreover, microservices are exposed to be reused freely [14]. However, orchestrating their consumption from an organization-wide perspective in a permissionless consumption approach calls for innovative governance. Indeed, allowing teams to change the boundaries of other microservices could lead to complexity in resolving and merging changes and new dependencies, violating the MSA's independence tenet [26]. Therefore, the question of allowing a reuse technique to bypass these fundamental tenets arises, stressing the need for more research in applying shared and coordinated reuse techniques.

Adopting an inner-sourcing approach guided by shared reuse ground rules was proposed to tackle this challenge [34]. However, practical evidence pointed out the complexity of harmonizing inner-sourcing ownership and microservice practices, mainly in allowing changes in a microservice not owned by the contributor, confirming the above assumption. Complying with reuse ground rules also jeopardizes a team's autonomy, as the team members cannot freely choose the microservices they use [34]. While reuse governance is needed to support organization-wide circular systems engineering, its achievement is complex because of microservices' distributed nature and management practices, which lead to the adoption of ineffective opportunistic and non-coordinated approaches [15]. Therefore, proposing innovative governance practices based on enforced organization-wide coordination is necessary.

2.4 Opportunistic Reuse

Software reuse relies on reusable parts and reuse opportunities. On the one hand, it leads to the need for design and development approaches that foster *development for reuse*. On the other hand, it leads to the need for approaches allowing developers to identify reuse opportunities and decide on using existing software parts, including retrieval processes, selection and adaptation of reusable artifacts, and reuse enactment, i.e., *development with reuse* [2].

Opportunistic development with reuse is unstructured, based on individual initiative, and relies on personal knowledge to make reuse decisions [6]. A typical example is software developers working on a project who encounter a requirement similar to one they solved in a previous project. Instead of designing and implementing a solution from scratch, they copy and paste relevant code snippets or modules from the previous project to speed up development. Current microservice reuse practices are opportunistic [20]. Indeed, a microservice is reused for a specific case based on the developer's personal choice and knowledge, which can lead to pass-by reuse opportunities [6]. This constitutes a significant barrier to the identification of reuse possibilities, thus limiting the expected microservices' reuse and circularity benefits.

The main limitation of this approach is that reuse focuses on solving immediate problems rather than considering broader reuse opportunities across products while introducing new challenges related to the creation of clones, depicting the impact of local design practices and poor team coordination. Additionally, quality concerns arise as, without proper evaluation processes, the reused component may introduce unexpected errors or inefficiencies into the reuse context, as the individual who decided to reuse it may not be aware of the context specificities. Besides, the reused part and its adaption to the new context may not be documented, preventing reusable identification, replication, and the projection of fixes [20]. Consequently, the challenge is no longer to create modular and reusable services but to identify the contexts in which they can be reused. Therefore, approaches supporting organization-wide reuse are mandatory for achieving sustainability goals.

3 PERSPECTIVES & MITIGATION STRATEGIES

This section introduces research and practice perspectives that have the potential to advance our understanding and methods of applying MDE to improve the design of sustainable MSAs. We built upon previous perspectives on marrying MSA and MDE [28] by incorporating sustainability concerns and integrating new knowledge. By exploring the perspectives proposed in this paper, we invite researchers and professionals to consider new paths to conciliate MSA tenets and MDE practices that can address the identified challenges and open up new avenues for investigation. These perspectives are designed to inspire fresh thinking and stimulate impactful research that bridges theory and practice. Table 1 outlines how MSA's aspects and challenges impact sustainability aspects, linking them to the perspectives for mitigation strategies we discuss below.

Table 1: Overview of Achieving Sustainability in MSA.

MSA Aspects	Challenges	Impact on Sustainability	Mitigation Strategies
Independence	Lack of Standards	Design	Bottom-up Modeling
Design	Local-driven Design	Design & Circularity	Real-time Modeling
Autonomy	Lack of Coordination	Design & Circularity	Systematic Analysis
Reuse	Opportunistic Reuse	Circularity	Systematic Reuse

3.1 Bottom-up Modeling

An end-to-end view of systems, including artifacts, activities, processes, and connections, should be explicit and formalized to support effective circularity and achievement of organization-wide sustainability goals [9]. Building such end-to-end knowledge involves adopting design practices that document microservices' internal and external aspects, comprising descriptions in functional and technological terms and explicitly identifying interactions between microservices. However, such a comprehensive MSA representation is challenged by the lack of standards and its independence and autonomy tenets.

Various modeling approaches exist in the MSA domain, and none seem dominant. Using separated models according to specific viewpoints is common practice [13], requiring mechanisms to combine the information necessary to analyze microservices holistically [23]. The ability to explore models takes over the unique

and separate representations aspect. Indeed, it is crucial to effectively extract and process data during analysis, regardless of how the information is represented [22]. Therefore, the first perspective is to explore **bottom-up modeling strategies that preserve teams' independence and autonomy while providing mechanisms to build organization-wide consistent systems views**. Microservices-based systems are modeled from a local and independent perspective that modeling approaches must consider. Besides, standards and tools allowing developers to model a 360-degree view of microservices architectures are still to be built [23]. In this context, exploring the transferability of reverse engineering model-driven approaches [30], collaborative modeling [32], and blended modeling [7] could be a great starting point. This perspective focuses on handling the lack of standard and shared approaches in MSA modeling.

3.2 Embracing Real-time Modeling

The main challenge when adopting systematic reuse is handling variability [12]. In MSA, variability encompasses managing variation at two levels: first, at the microservice individual level, which needs to consider concomitant versions. Second, there is a variation in the arrangements of microservices, as a microservice can participate in various compositions, which might be created on the fly during deployment.

Therefore, the second perspective is to explore **models at runtime approaches**. Such approaches help keep microservices representation up-to-date by identifying version changes and deployment interactions resulting from dynamic microservices arrangements. Documenting systems architectures is challenging because documentation must be kept up-to-date, especially in fast-paced development approaches like MSA [14]. This necessity underscores the risk of making decisions based on outdated data, which can lead to inappropriate reuse identification. This perspective aims to suggest modeling approaches that handle the rapid evolution of MSA and document variability to support systematic reuse methodologies, tackling the challenge of local-driven design.

3.3 Systematic Sustainability Analysis in MSA

Modeling sustainable systems starts by understanding and making explicit goals and constraints, analyzing sustainability from social, economic, environmental, and technical perspectives [18]. These perspectives form evaluation objectives, which assess a specific sustainability dimension throughout sustainability quality requirements measured using evaluation criteria aligned with stakeholder concerns. They underscore the relevance of unveiling the interplay between the assessed aspects, as sustainability concerns can support each other or be conflictual. Making these relations visible through adequate representation allows stakeholders to reach agreements, conciliating the diversity of sustainability goals and their positive or negative influences on each other.

Therefore, the third perspective is to explore **the applicability of systematic analysis frameworks in a granular and coordinated way**, i.e., allowing to derive local microservices sustainability objectives and coordinate them among teams. MDE meta-modeling practices and transformation mechanisms could support such systematic approaches and enhance team coordination throughout automatic model processing [3]. Automatic model processing could identify and link aspects in a multi-model space [23, 25, 27] to support sustainability analysis and alignment. Besides, automatic transformations could derive specific models from a cross-organization one [5] to coordinate local sustainability strategies enactment, tackling the lack of coordination between teams.

3.4 Embracing Systematic Reuse

Enabling sustainability benefits by adopting MSA calls for facilitating microservices' reuse on larger scales and optimizing reuse practices to meet sustainability goals. In this context, systematic reuse represents a way to achieve circularity in microservices composition as it can streamline and generalize "development with reuse" in MSA. Such a process relies on *abstraction* and *automation* to support matching reusable artifacts and reuse opportunities [17]. However, the systematic reuse process that supports MSA tenets has yet to be built [1].

Implementing systematic reuse involves centralized processes and important initial investment, which may not comply with MSA independence and autonomy tenets [6, 20]. Fostering systematic reuse without violating MSA tenets calls for aligning rigorous governance models with microservices management, interfering as little as possible in team dynamics, and fostering automatic processes to limit costs. Consequently, accommodating reuse governance and the independence and autonomy of teams calls for comprehensive solutions at the organizational level [26]. These aspects have yet to be explored in actual development with reuse approaches applied to MSA. Managing reuse at the business process level helps cope with ownership and governance issues, and make explicit functional dependencies between services. Sun et al. [33] applied BPM and MDE to implement a systematic microservices reuse approach based on business processes. However, their approach overlooks sustainability, not assessing the impact of additional consumers on resource consumption and systems performance.

Therefore, the fourth perspective is to explore **how existing MDE and business process modeling approaches can collaborate to support systematic development with reuse in MSA**, i.e., how microservices reuse can be supported by models that consider organization and sustainability objectives, concerns, and constraints. Thus, this perspective tackles the limitations caused by opportunistic reuse.

4 POTENTIAL ISSUES OF MDE-BASED MSA

Applying MDE to support sustainable MSA comes with several research challenges. A primary issue is model management in collaborative contexts [8]. For instance, handling multi-view modeling, model conflict detection, and collaboration supported by automation may impact the feasibility of MDE in the MSA context. Indeed, collaborative modeling for MSA remains limited to internal team concerns and roles, overlooking external collaborations [29]. Besides, handling runtime adaptation complexity is a foundation challenge in MDE [4], which may be exacerbated by MSA complexity. Similarly, sustainability and stability criteria must be integrated when analyzing and adapting systems at runtime, which may be challenging in present MDE practices [11, 31]. Moreover, adopting

an MDE approach can be costly due to the need for specific training and tooling, which may pose a barrier for organizations [4].

5 CONCLUSION

This paper discussed the weaknesses of current MSA practices in achieving sustainable MSA-based systems and presented perspectives on developing MDE-based approaches that tackle these weaknesses. Such approaches must leverage the granularity in design and operation fostered by MSA while preserving the independence and autonomy tenets that are the motivational elements for various organizations adopting this paradigm [37]. Even if MDE practices seem promising, we must extend our theoretical and practical knowledge and continuously reflect on and stay critical about using cross-team, centralized, and top-down modeling approaches in MSA. Therefore, we plan to explore new avenues for implementing MDE practices under these constraints. Currently, we are working on bottom-up strategies to build cross-team views of systems supported by mining mechanisms [23]. We aim to provide a limitedly intrusive approach to team practices, leveraging their current documentation methods to build unified views of microservices systems that can be explored top-down to generate stakeholder-related views. Besides, we are working on designing sustainability profiles and metrics to support sustainability-driven decisions in MSA.

6 ACKNOWLEDGEMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), 06351.

REFERENCES

[1] Wesley KG Assunção, Jacob Krüger, and Willian DF Mendonça. 2020. Variability management meets microservices: six challenges of re-engineering microservice-based webshops. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A*. 1–6.

[2] Jose L Barros-Justo, Fabiane BV Benitti, and Santiago Matalonga. 2019. Trends in software reuse research: A tertiary study. *Computer Standards & Interfaces* 66 (2019), 103352.

[3] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-driven software engineering in practice*. Morgan & Claypool Publishers.

[4] Antonio Bucchiarone, Jordi Cabot, Richard F Paige, and Alfonso Pierantonio. 2020. Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling* 19 (2020), 5–13.

[5] Loli Burgueno, Jordi Cabot, Shuai Li, and Sébastien Gérard. 2022. A generic LSTM neural network architecture to infer heterogeneous model transformations. *Software and Systems Modeling* 21, 1 (2022), 139–156.

[6] Rafael Capilla, Barbara Gallina, Carlos Cetina, and John Favaro. 2019. Opportunities for software reuse in an uncertain world: From past to emerging trends. *Journal of Software: Evolution and process* 31, 8 (2019), e2217.

[7] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. 2019. Blended modelling-what, why and how. In *ACM/IEEE 22nd Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 425–430.

[8] Istvan David, Kousar Aslam, Ivano Malavolta, and Patricia Lago. 2023. Collaborative Model-Driven Software Engineering—A systematic survey of practices and needs in industry. *Journal of Systems and Software* 199 (2023), 111626.

[9] Istvan David, Dominik Bork, and Gerti Kappel. 2024. Circular systems engineering. *Software and Systems Modeling* (2024), 1–15.

[10] Nicola Dragoni, Ivan Lanese, Stephan Thordal Larsen, Manuel Mazzara, Ruslan Mustafin, and Larisa Safina. 2018. Microservices: How to make your application scale. In *Perspectives of System Informatics: 11th International Andrei P. Ershov Informatics Conference*. Springer, 95–104.

[11] Iffat Fatima and Patricia Lago. 2024. Software Architecture Assessment for Sustainability: A Case Study. In *Software Architecture*. Springer, Cham.

[12] Matthias Galster and Paris Avgeriou. 2013. Variability in Web services. In *Systems and Software Variability Management*. Springer, 269–278.

[13] Saverio Giallorenzo, Fabrizio Montesi, Marco Peressotti, Florian Rademacher, and Sabine Sachweh. 2021. Jolie and LEMMA: model-driven engineering and programming languages meet on microservices. In *International Conference on Coordination Languages and Models*. Springer, 276–284.

[14] Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. 2017. Performance Engineering for Microservices: Research Challenges and Directions. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion* (L'Aquila, Italy). ACM, 223–226. https://doi.org/10.1145/3053600.3053653

[15] Reid Holmes and Robert J Walker. 2013. Systematizing pragmatic software reuse. *ACM Trans. Softw. Eng. Methodol.* 21, 4 (2013), 1–44.

[16] Lalita J. Jagadeesan and Veena B. Mendiratta. 2020. When Failure is (Not) an Option: Reliability Models for Microservices Architectures. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 19–24. https://doi.org/10.1109/ISSREW51248.2020.00031

[17] Charles W Krueger. 1992. Software reuse. *ACM Computing Surveys (CSUR)* 24, 2 (1992), 131–183.

[18] Patricia Lago, Sedef Akinli Koçak, Ivica Crnkovic, and Birgit Penzenstadler. 2015. Framing sustainability as a property of software quality. *Commun. ACM* 58, 10 (2015), 70–78.

[19] Olivier Le Goaer, Adel Noureddine, Franck Barbier, Romain Rouvoy, and Florence Maraninchi. 2021. Vers des Logiciels Éco-responsables. (2021).

[20] Niko Mäkitalo, Antero Taivalsaari, Arto Kiviluoto, Tommi Mikkonen, and Rafael Capilla. 2020. On opportunistic software reuse. *Computing* 102 (2020), 2385–2408.

[21] Manuel Mazzara, Antonio Bucchiarone, Nicola Dragoni, and Victor Rivera. 2020. Size matters: Microservices research and applications. In *Microservices*. Springer, 29–42.

[22] Gabriel Morais, Mehdi Adda, Hiba Hadder, and Dominik Bork. 2023. x 2OMSAC-An Ontology Population Framework for the Ontology of Microservices Architecture Concepts. In *World Conference on Information Systems and Technologies*. Springer, 263–274.

[23] Gabriel Morais, Dominik Bork, and Mehdi Adda. 2021. Towards an Ontology-driven Approach to Model and Analyze Microservices Architectures. In *Proceedings of the 13th Int. Conf. on Management of Digital EcoSystems*. 79–86.

[24] Sam Newman. 2021. *Building microservices*. " O'Reilly Media, Inc.".

[25] James Pontes Miranda, Hugo Bruneliere, Massimo Tisi, and Gerson Sunyé. 2024. Towards the Integration Support for Machine Learning of Inter-Model Relations in Model Views. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*. 1304–1306.

[26] Florian Rademacher, Jonas Sorgalla, and Sabine Sachweh. 2018. Challenges of domain-driven microservice design: A model-driven perspective. *IEEE Software* 35, 3 (2018), 36–43.

[27] Florian Rademacher, Jonas Sorgalla, Sabine Sachweh, and Albert Zündorf. 2019. Viewpoint-Specific Model-Driven Microservice Development with Interlinked Modeling Languages. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. 57–5709. https://doi.org/10.1109/SOSE.2019.00018

[28] Florian Rademacher, Jonas Sorgalla, Philip Nils Wizenty, Sabine Sachweh, and Albert Zündorf. 2018. Microservice architecture and model-driven development: Yet singles, soon married (?). In *Proceedings of the 19th International Conference on Agile Software Development: Companion*. 1–5.

[29] Florian Rademacher, Philip Wizenty, Jonas Sorgalla, Sabine Sachweh, and Albert Zündorf. 2024. Model-Driven Engineering of Microservice Architectures—The LEMMA Approach. In *Ernst Denert Award for Software Engineering 2022: Practice Meets Foundations*. Springer Nature Switzerland Cham, 105–147.

[30] Claudia Raibulet, Francesca Arcelli Fontana, and Marco Zanoni. 2017. Model-Driven Reverse Engineering Approaches: A Systematic Literature Review. *IEEE Access* 5 (2017), 14516–14542. https://doi.org/10.1109/ACCESS.2017.2733518

[31] Maria Salama, Rami Bahsoon, and Patricia Lago. 2019. Stability in software engineering: Survey of the state-of-the-art and research directions. *IEEE Transactions on Software Engineering* 47, 7 (2019), 1468–1510.

[32] Jonas Sorgalla, Florian Rademacher, Sabine Sachweh, and Albert Zündorf. 2018. On collaborative model-driven development of microservices. In *Federation of Int. Conf. on Software Technologies: Applications and Foundations*. Springer, 596–603.

[33] Chang-ai Sun, Rowan Rossing, Marco Sinnema, Pavel Bulanov, and Marco Aiello. 2010. Modeling and managing the variability of Web service-based systems. *Journal of Systems and Software* 83, 3 (2010), 502–516.

[34] Muhammad Usman, Deepika Badampudi, Chris Smith, and Himansu Nayak. 2022. An Ecosystem for the Large-Scale Reuse of Microservices in a Cloud-Native Context. *IEEE Software* 39, 05 (2022), 68–75.

[35] F. H. Vera-Rivera, C. Gaona, and H. Astudillo. 2021. Defining and measuring microservice granularity—a literature overview. *PeerJ Computer Science* 7 (2021), e695. https://doi.org/10.7717/peerj-cs.695

[36] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle, and Gastón Márquez. 2021. Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software* 182 (2021), 111061.

[37] Xin Zhou, Shanshan Li, Lingli Cao, He Zhang, Zijia Jia, Chenxing Zhong, Zhihao Shan, and Muhammad Ali Babar. 2023. Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. *Journal of Systems and Software* 195 (2023), 111521.