2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) | 978-1-6654-0337-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/ASE51524.2021.9678830

Semi-automated Cross-Component Issue Management and Impact Analysis

Sandro Speth
Institute of Software Engineering
University of Stuttgart
Stuttgart, Germany
sandro.speth@iste.uni-stuttgart.de

***Abstract*—Despite microservices and other component-based architecture styles being state of the art in research for many years by now, issue management across the boundaries of a single component is still challenging. Components that were developed independently and can be used independently are joined together in the overall architecture, which results in dependencies between those components. Due to these dependencies, bugs can result that propagate along the call chains through the architecture. Other types of issues, such as the violation of non-functional quality properties, can also impact other components. However, traditional issue management systems end at the boundaries of a component, making tracking of issues across different components time-consuming and error-prone. Therefore, a need for automation arises for cross-component issue management, which automatically puts issues of the independent components in the correct mutual context, creating new cross-component issues and syncing cross-component issues between different components. This automation could enable developers to manage issues across components as efficiently as possible and increases the system's quality. To solve this problem, we propose an initial approach for semi-automated cross-component issue management in conjunction with service-level objectives based on our Gropius system. For example, relationships between issues of the same or different components can be predicted using classification to identify dependencies of issues across component boundaries. In addition, we are developing a system to model, monitor and alert service-level objectives. Based on this, the impact of such quality violations on the overall system and the business process will be analysed and explained through cross-component issues.**

***Index Terms*—Automated Issue Management, Component-based Architectures, Cross-Component Issues, SLO Monitoring**

I. Problem

Component-based architectural styles, such as microservices, have grown in popularity in recent years due to their many advantages. In this context, independently developed and deployable components are assembled into larger architectures [1]. However, although the individual components may be resilient, this no longer applies to the overall architecture due to the emerging dependencies between the individual components. Thus, bugs or other types of issues can arise that propagate across components and their interfaces through the architecture, thereby affecting multiple components. In the worst case, a bug in an interface of one component can cause the cascading failure of one or more other components that consume the initially affected component [2].

Although systems with such modern architectural styles involve multiple teams and components, traditional issue management ends at the boundaries of a single component. In addition, each development team is free to choose its technology stack [3], [4] and thus components often use different issue management system vendors. Identifying the root cause of an issue, tracking issues across multiple components, and identifying relationships between issues of two components is thus made difficult and is both time-consuming and error-prone [5], [6]. In particular, issues that affect several components have to be synchronised between the issue management systems of the respective components at great effort [7]. We call such kind of issues and issues that result in other issues for depending components as *cross-component issues*. To counteract this, we have developed *Gropius*[1] [8], which acts as a wrapper over established issue management systems and thus enables the management of such cross-component issues. The architecture of the system and the issues are displayed together in a graphical modelling language similar to a UML component diagram [9]. This makes it easier to identify dependencies between issues of different components. Changes to cross-component issues are propagated and synchronised by Gropius to the underlying issue management systems.

However, the detection of relationships between issues of different components remains a manual process that is further complicated by misclassification of the issue type [10], i.e. bug report or feature request. To improve this manual process, AI can be used to enable automation. For example, the type of an issue can be recognised by classification. Furthermore, predictions can be made when creating a new issue to determine which existing issues of the same component or components dependent on it could be related to the newly created issue. Such prediction allows to quickly identify dependencies between issues of multiple components without having to check dozens of issues from multiple issue management systems manually. As a result, issues can be put into the correct context and contain as much crucial information as possible that is relevant for the fix of the issue [11].

Another factor is the explanation of the impact of quality violations on the system and its business processes. The origin

[1] https://github.com/ccims/ccims-backend-gql and https://github.com/ccims/ccims-frontend

978-1-6654-0337-5/21/$31.00 ©2021 IEEE
DOI 10.1109/ASE51524.2021.00108

of a service-level objective (SLO) violation may not necessarily be the component itself for which the SLO was violated but may have evolved through various preceding components. In self-adapting systems, SLO violations may only be temporary due to self-healing. Nevertheless, frequently recurring violations should be recognised and explained. An explanation of such failure sources can be integrated into an established part of a development process through targeted monitoring of SLOs and automatic creation of cross-component issues. Therefore, we present a tool and concept with which SLOs can be modelled and monitored, and their violations can be alerted as issues. Based on this, automated analyses can be carried out to identify correlations between SLO violations and their effects on business processes.

II. Related Work

Related work for our approach can be classified into (1) cross-component issue management, (2) SLO monitoring and alerting, and (3) impact analysis of SLO violations on architectures and business processes.

As for scientific papers on cross-component issues, only our own on Gropius, our Cross-Component Issue Metamodel, and the Gropius Cross-Component Issue Modelling Language can be found. Additionally, some issue management system plugins, such as the Multi-Project Picker for Atlassian, attempt to synchronise issues between multiple components. However, these plugins are limited to single vendors and do not include semantic links from issues to issues of other components. Several works classify issues in an automated way (e.g. [10], [12]–[14]). These approaches usually only focus on the question of whether the issue is a bug or not. Classification into more than two issue types and whether it affects an interface of a component is not considered. Work on predicting relationships between issues, e.g. [15]–[17], only considers duplicate detection rather than other relevant relationships, such as *depends on*.

The research area of service-level objectives for component-based architectures covers a broad area, which includes, e.g. Web Service-Level Agreements (WSLA) [18]. The monitoring of SLOs was considered, for example, in SALMon by Ameller et al. [19], whereby SALMon does not consider serverless applications or components deployed in Kubernetes (K8), which is becoming increasingly popular in modern architectures. Nastic et al. [20] on the one hand, already consider cloud applications, but on the other hand, also do not focus on K8 and serverless applications. In particular, while these approaches monitored SLOs, they do not produce detailed analyses and explanations of how SLO violations occurred. Therefore, developers still have to search for the root cause of the problem producting the occuring SLO violation.

There is some work regarding root cause analysis concerning SLO violations. An analysis of composite services to find out which dependency caused the SLO violation was in [21], [22]. While they look at the decomposed level of services, they do not consider the impact of higher levels, such as the business processes. The MYCROLYZE framework [23] recovers the dynamic architecture of a system using tracing. Furthermore, it enables the mapping of business processes to the recovered architecture to recover correlations between the architecture and business processes. To reduce the cost of recovery, Hanemann et al. take the impact of resource failure on SLO violations into consideration [24]. However, they do not cover violations propagating across the architecture.

III. Proposed Approach

This section provides a brief overview of our approach for semi-automated cross-component issue management and impact analysis. Figure 1 depicts a high-level domain model containing the essential parts of our approach.

In traditional issue management, an issue has exactly one location defined by the issue management system in which the issue is managed. For modern component-based architectures, however, issues often need to be managed across the boundaries of the individual components and, therefore, their issue management systems. To make this possible, we have introduced Gropius [8], which acts as a wrapper above existing issue management systems. Gropius implements the Cross-Component Issue Metamodel [9], which extends an issue so that it can affect more than one location. A location consists of a component and can optionally be specified for the provided interface of the component. Issues that affect several components can thus be synchronised by Gropius between the underlying issue management systems. Using the graphical Gropius Cross-Component Modelling Language (GML)-[9], cross-component issues are represented together with the architecture of the system in a UML component diagram-like graph. The architecture consists of components that can provide interfaces or consume interfaces of other components. Issues can be related to issues of other components, for example, in a *depends on* relationship. The GML's graphical concrete syntax illustrates such relationships and thus enables developers to efficiently track issues across the boundaries of individual components.

However, identifying such relationships and adding them to the issues is both time-consuming and error-prone. To tackle this issue, we propose to predict such issue relationships employing machine learning automatically. An *Issue Relationship Predictor* could identify possible relationships by classifying pairs of issues for each relationship. Such a machine learning model can be based on BERT [25]. Predicting the relationship between issues allows automatically putting cross-component issues in the proper context. Furthermore, such a predictor is not restricted to existing issues but enables recommendations of issue relationships when creating a new cross-component issue. As a result, possible dependencies to already existing issues could easily be identified while creating a new issue. Additionally, the type of an issue and whether the issue affects a component's interface can be classified automatically during creation. This prevents misclassification of issues, as is often the case in manual procedures [10], which can quickly lead to duplicates or the overlooking of a bug, for example.

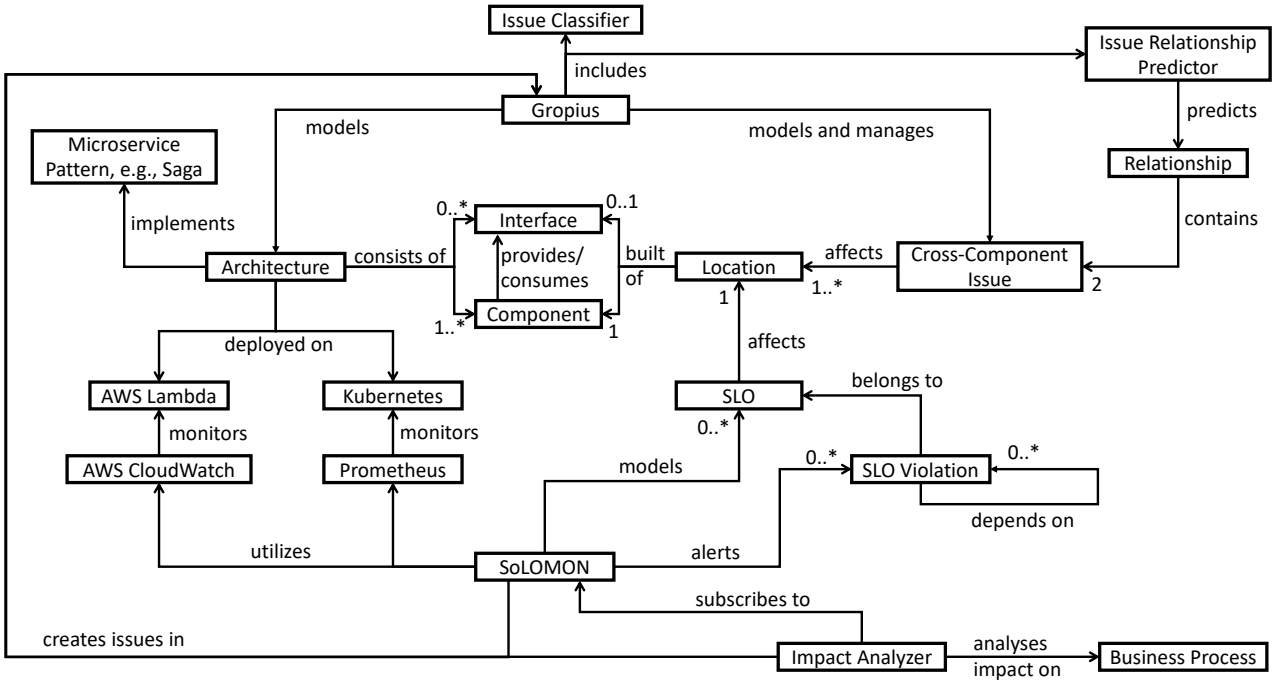

Fig. 1. Overview of our approach for semi-automated cross-component issue management and impact analysis.

Nevertheless, putting existing issues in the correct context and manually identifying new issues is no longer sufficient for self-adaptive systems, such as many microservice architectures. Self-adaptation makes it challenging to identify Quality of Service (QoS) issues. In particular, by the time a DevOps team analyses the system, the QoS issue may be temporarily fixed but may recur, possibly causing frequent violations of the affected component's SLOs. As a result, automatic detection of such SLO violations and generating meaningful explanations of how the violation arose and whether it may be self-recovering through self-adaptivity is needed. Therefore, our approach introduces *SoLOMON*[2], a tool for modelling and monitoring SLOs and automatically generating alerts and explanations of SLO violations as cross-component issues in Gropius. To model SLOs in an issue, cross-component issues can have non-functional properties for the issues' contents, for example, a maximum response time for a new endpoint. The architectures' components to be monitored can be deployed in a Kubernetes (K8) cluster or be serverless, for example, in AWS Lambda or Microsoft Azure Functions. SoLOMON transforms the modelled service-level objectives for each component into rules for a Prometheus Blackbox Exporter or AWS CloudWatch, which then alert SoLOMON of SLO violations. Then, SoLOMON offers these alerts to a subscriber for further processing. Furthermore, SoLOMON creates a cross-component issue for Gropius containing an explanation based on the information provided by the Prometheus Blackbox Exporter and AWS CloudWatch, taking dependencies between SLO violations into consideration. For a more fine-grained SLO level, i.e. on endpoint basis, tracing tools like Jaeger, Zipkin, and OpenTracing could be used within SoLOMON.

As a subscriber for SoLOMON, we introduce a tool to analyse the impact of SLO violations on the system's business process. This *Impact Analyser* implements a modelling language to model the architecture next to its business process. Furthermore, microservice patterns, such as the Saga pattern [26], can be modelled, too. In case of an SLO violation, the Impact Analyser can identify the impact on the business process by following modelled links from the architecture to the system's business process. Especially if the architecture implements a Microservice pattern having an effect on the data consistency such patterns might have an influence on the impact. For example, if an architecture implements the Saga pattern, SLO violations might impact the business process silently as the Saga rolled back, resulting in a rollback of the business process. As a result, the business process could seem not to be triggered at all. Even if a problem is recognized it might be hard to find the exact root cause of the components participating in the saga. After automatically identifying the affected components and steps of the business process, the Impact Analyser can report the impact of an SLO violation as cross-component issues for the overall project in Gropius. The Issue Relationship Predictor can then determine possible relationships between several SLO violation alerts making the overall failure of the system more transparent.

[2]https://github.com/ccims/solomon

IV. Expected Contributions

We will have three main contributions trying to help developers managing cross-component issues in an efficient and effective manner: (1) Our first contribution will be Gropius which is a system to model and integrated manage cross-component issues. (2) As a second contribution, we automatically identify the correct context for cross-component issues, i.e. issue type (bug report or feature request), issue location, and relationships to other issues of the same or different components. (3) The third contribution tries to automatically create cross-component issues containing explanations of SLO violations and their impact on the overall architecture and its business process. The system creating these explanations should keep Microservice patterns and self-adaptivity in mind.

V. Current Status

Currently, we are at the first phase of our work which consists of building a concept and tool support for integrated cross-component issue management. The ideas in this paper reflect in total a set of five work packages describing our planned steps how we try to realize our approach of a semi-automated cross-component issue management and impact analysis. The work packages are briefly outlined in the following.

a) Integrated Cross-Component Issue Management: This work package extends the metamodel of issues to enable cross-component issues, i.e. issues affecting more than one component or resulting in other issues of the same or dependent components. Furthermore, it contains the development of a graphical modelling language for cross-component issues and a tool to manage cross-component issues. The cross-component issue metamodel and its graphical modelling language should be implemented in a tool which integrates the existing issue management systems and maps the cross-component issues' features to them.

b) Automated Cross-Component Issue Context Identification: In this work package, we try to automatically identify the correct issue type, whether an issue affects a component's interface and relationships between issues of the same or different components by means of artificial intelligence.

c) SLO Modelling, Monitoring, and Alerting: For this work package, we plan to build a system allowing us to model and monitor SLOs of component-based architectures. SLO violations should be automatically alerted via a subscriber approach. One of the subscribers can create explanations of these SLO violations as cross-component issues in Gropius.

d) Automated Impact Analysis: This work package targets the automatic analysis of SLO violations on the overall architecture and tries to explain them. In particular, if the architecture implements microservice patterns [26], e.g. the Saga pattern, or if the system is self-adaptive, the root cause could already be solved, but recurring making explanations more difficult. The system should be able to include such information to its explanations.

e) Evaluation: In this work package, we evaluate our approach using Goal-Question Metric, a user study, and experiments as described in section VI.

VI. Plan for Evaluation and Validation

We plan to evaluate our approach and its expected benefits based on a Goal-Question-Metric [27] approach with a user study in case we can find an open-source or industry project to cooperate. Additionally, the approach could also be evaluated with experiments using modern microservice reference architectures, such as the T2-project[3] which is currently developed by us and based on the TeaStore [28]. It implements the microservice Saga pattern [26]. One of its purposes is to benchmark SLO violations. The issue relationship prediction can be evaluated by validating the classification models with a large test set and calculating the F1-score and accuracy.

References

[1] C. Szyperski, D. Gruntz, and S. Murer, *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[2] J. Soldani and A. Brogi, "Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey," *arXiv preprint arXiv:2105.12378*, 2021.

[3] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, 2015. [Online]. Available: http://my.safaribooksonline.com/9781491950357

[4] M. Nygard, *Release It!: Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2007.

[5] Á. Brandón, M. Solé, A. Huélamo, D. Solans, M. S. Pérez, and V. Muntés-Mulero, "Graph-based root cause analysis for service-oriented and microservice architectures," *Journal of Systems and Software*, vol. 159, 2020.

[6] S. Mahmood, M. Niazi, and A. Hussain, "Identifying the challenges for managing component-based development in global software development: Preliminary results," in *2015 Science and Information Conference (SAI)*. IEEE, 2015, pp. 933–938.

[7] S. Speth, "Issue management for multi-project, multi-team microservice architectures," Master's thesis, University of Stuttgart, 2019.

[8] S. Speth, U. Breitenbücher, and S. Becker, "Gropius—a tool for managing cross-component issues," in *Communications in Computer and Information Science*, vol. 1269, Springer. Springer, 2020, pp. 82–94.

[9] S. Speth, S. Becker, and U. Breitenbücher, "Cross-component issue metamodel and modelling language," in *Proceedings of the 11th International Conference on Cloud Computing and Services Science (CLOSER 2021)*, INSTICC. SciTePress, May 2021, pp. 304–311.

[10] S. Herbold, A. Trautsch, and F. Trautsch, "On the feasibility of automated prediction of bug and non-bug issues," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5333–5369, 2020.

[11] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. ACM, 2008, pp. 308–318.

[12] N. K. Nagwani and S. Verma, "Clubas: An algorithm and java based tool for software bug classification using bug attributes similarities," 2012.

[13] ——, "Ml-clubas: A multi label bug classification algorithm," 2012.

[14] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement? a text-based approach to classify change requests," in *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, 2008, pp. 304–318.

[15] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 183–192.

[16] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2012, pp. 70–79.

[3] https://github.com/t2-project

[17] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Software Engineering*, vol. 21, no. 2, pp. 368–410, 2016.
[18] A. Keller and H. Ludwig, "The wsla framework: Specifying and monitoring service level agreements for web services," *Journal of Network and Systems Management*, vol. 11, no. 1, pp. 57–81, 2003.
[19] D. Ameller and X. Franch, "Service level agreement monitor (salmon)," in *Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008)*. IEEE, 2008, pp. 224–227.
[20] S. Nastic, A. Morichetta, T. Pusztai, S. Dustdar, X. Ding, D. Vij, and Y. Xiong, "Sloc: Service level objectives for next generation cloud computing," *IEEE Internet Computing*, vol. 24, no. 3, pp. 39–50, 2020.
[21] X. Huang, S. Zou, W. Wang, and S. Cheng, "Mdfm: Multi-domain fault management for internet services," in *IFIP/IEEE International Conference on Management of Multimedia Networks and Services*. Springer, 2005, pp. 121–132.
[22] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 492–502.
[23] M. Kleehaus, Ö. Uludağ, P. Schäfer, and F. Matthes, "Microlyze: a framework for recovering the software architecture in microservice-based environments," in *International Conference on Advanced Information Systems Engineering*. Springer, 2018, pp. 148–162.
[24] A. Hanemann, D. Schmitz, and M. Sailer, "A framework for failure impact analysis and recovery with respect to service level agreements," in *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, vol. 2. IEEE, 2005, pp. 49–56.
[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[26] C. Richardson, *Microservices patterns: with examples in Java*. Simon and Schuster, 2018.
[27] V. R. B. G. Caldiera and H. D. Rombach, "The goal question metric approach," *Encyclopedia of software engineering*, pp. 528–532, 1994.
[28] J. von Kistowski *et al.*, "Teastore: A micro-service reference application for benchmarking, modeling and resource management research," in *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2018, pp. 223–236.