Engineering Reliable Hybrid Quantum Software: An Architectural-driven Approach

Max Scheerer[1], Jonas Klamroth[1] and Oliver Denninger[1]

[1]FZI Research Center for Information Technology, Karlsruhe, Germany

Abstract

Engineering quantum software typically involves the implementation of well-known quantum algorithms as quantum software components and their embedding into a hybrid software architecture – executed on quantum hardware as well as classical hardware. While error models of modern microprocessors are quite similar and error mitigation is handled by compilers, this is not true for current quantum processors. The reliability of quantum software depends heavily on the quantum hardware used for execution. Quantum software engineers have to consider deployment decisions to estimate the reliability impact of quantum software components to the overall software architecture. We propose a model-based approach for the automated analysis of hybrid quantum software at design-time, taking into account the aforementioned reliability challenges. By extending the Palladio framework for modeling and simulating software architectures, we built upon well-established tooling. We support quantum software engineers making design decisions, by automatically exploring the design space including the impact of quantum hardware.

Keywords

Quantum Architectural Description Language, Software Architecture, Quantum Computing, Hybrid Quantum Application

1. Introduction

With the availability of small-scale quantum computers, *Quantum Computing* (QC) and *Quantum Algorithms* gained a lot attraction. QC promises to speed up computationally intensive algorithms. For instance, Shor's algorithm [1] achieves an exponential speedup in factorizing numbers with potential impact to RSA-based encryption. Implications of QC are not limited to cyber security, but apply to commercial applications in, e.g., chemistry, finance and manufacturing [2]. In the current *Noisy Intermediate-Scale Quantum* (NISQ) era, quantum computers exhibit short decoherence times (i.e., the time to maintain a stable quantum state) and limited accuracy of gate operations. Hence, the depth of a quantum circuit must be kept small to avoid erroneous execution [3]. This enforces quantum software engineers to embed small quantum software components into software architectures designed for classical hardware, actually transforming them into hybrid quantum software architectures [4].

According to Salm et al. [5], there are two main challenges when developing QC software for NISQ devices: (i) there is a variety of available quantum computers with different physical properties (e.g., number of qubits or decoherence times) to which a quantum algorithm can

2nd Quantum Software Engineering and Technology Workshop (QSET'21), 2021

scheerer@fzi.de (M. Scheerer); klamroth@fzi.de (J. Klamroth); denninger@fzi.de (O. Denninger)

be deployed to and (ii) implementations of quantum algorithms are tightly coupled to vendor specific quantum computers and their *Software Development Kit* (SDK) so that they cannot be (efficiently) executed on all quantum computers. Salm et al. developed the *NISQ Analyzer* to calculate the fit between a quantum algorithm implementation and different quantum processors.

In this paper, we proposed to extend the idea behind *NISQ Analyzer* to exploring the overall design space of hybrid quantum software architectures. We propose to utilize reliability estimations for quantum software components on different quantum processors (deployments) to guide architectural design decisions. Our approach extends an *Architectural Description Language* (ADL) known as *Palladio Component Model* (PCM) [6] to a *quantum Architectural Description Language* (qADL), as envisioned by Zhao [7]. Based on the qADL models, we discuss how to predict reliability properties of hybrid quantum software architectures. We built upon a reliability prediction approach for PCM models [8], by integrating the *Estimated Success Probability* (ESP) metric [9, 10, 11] to quantify the reliability of a quantum software component w.r.t. its deployment. Finally, we discuss the integration in the PCM-based *PerOpteryx* tool [12] to automatically explore the design space.

The contribution of the paper is a novel approach to support software engineers in designing hybrid quantum software. The concepts are expected to inspire new research directions for modeling and analyzing hybrid quantum software at design-time.

2. Preliminaries

In this section, we briefly discuss concepts of the *Palladio Component Model* (PCM) and present an error model used to estimate the reliability of quantum algorithms executed on quantum computers.

2.1. Palladio Component Model

The PCM [6] is an ADL for component-based software architectures and encompasses a set of models that are divided into *structural*, *behavioral* and *deployment* viewpoints. The structural view point consists of a *Repository Model* to describe all available components and their provided and required interfaces. The provided and required interface specification induces a dependency structure of the components, i.e., components that require a certain interface are dependent on the components that provide it. The *System Model* specifies the instantiated components of the repository model and forms the runtime model. Behavioral view point models enable domain experts to describe usage scenarios of the system, e.g., the request rate of the system. The intra-component behavior of a component can be modeled as control flow consisting of sequences of abstract actions such as internal actions, branch actions and external component calls. The deployment view point comprises models for (i) describing the resource environment (i.e., the available servers) and (ii) the deployment structure (i.e., the allocation of instantiated components on servers). Figure 1 depicts an example PCM model.

The PCM is complemented by various simulation tools that enable the prediction of non-functional quality attributes (e.g., performance or reliability) and together form the Palladio framework. One of these tools is known as *PCM-Rel* [8] for predicting reliability properties of a PCM model, i.e., the success probability of a request to the system. In a nutshell, *PCM-Rel*

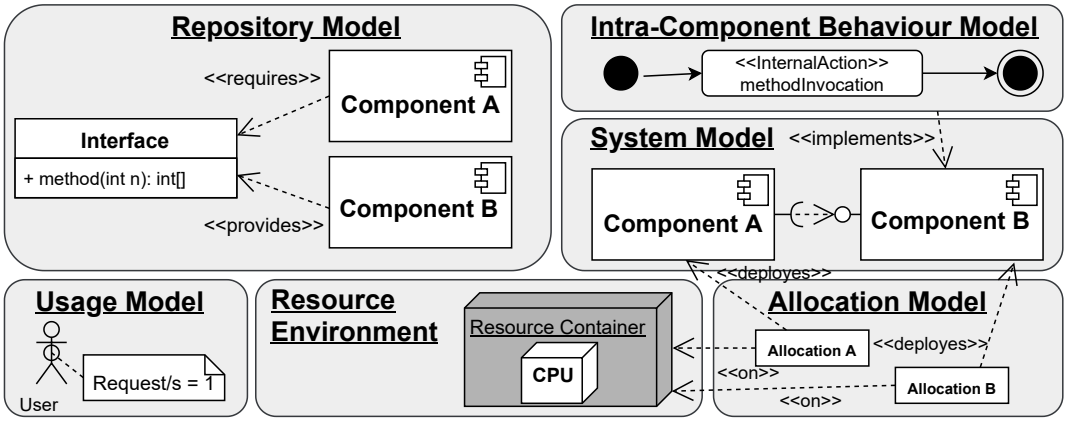

Figure 1: PCM example model

generates (based on the structural, behavioral and deployment information as well as software, network and hardware failure annotations provided by a modelled PCM instance) an absorbing *Discrete-Time Markov Chain* (DTMC). The DTMC is evaluated to predict the system's success probability.

Finally, the Palladio framework provides an automated design space exploration tool, called *PerOpteryx* [12]. Each software architecture is associated with several degrees of freedom or design decisions, e.g., the allocation of a component to server A or B; selection of component implementations providing the same interface. The set of design decisions span the design space $\mathcal{D}$ that includes architectural candidates. Each candidate in $\mathcal{D}$ has a different effect on the quality attributes. Software architects must explore the design space in order to identify the candidate that satisfies the non-functional requirements. For complex software systems, however, $\mathcal{D}$ is too large to be explored manually. Therefore, PerOpteryx provides an approach based on evolutionary algorithms which explores $\mathcal{D}$ automatically. The result of the analyses is a set of PCM models that forms the set of Pareto-optimal candidates w.r.t. the considered quality attributes. A software engineer can select from the Pareto-optimal set the candidate that best balances the predicted quality attributes.

2.2. Error Model used in Quantum Computing

Nishio et al. [9] categorize quantum errors into three groups, namely *Single-Qubit Gate-*, *Bi-Qubit Gate-* as well as *State Preparation* and *Measurement* errors. Single-qubit gate errors determine unitary bit (e.g., $|0\rangle \rightarrow |1\rangle$) or phase flips (e.g., $\alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \alpha\,|0\rangle - \beta\,|1\rangle$, $\alpha, \beta \in \mathbb{C}$). Bi-qubit gate errors are similar to single-qubit gate errors by flipping values of one or both qubits. State preparation and measurement errors may occur during initialisation of the qubits or due to readout errors of the qubits.

Each of the grouped quantum errors is associated with an error rate such that the *Estimated*

Success Probability can be calculated [9, 10, 11]:

$$ESP = \prod_{i=1}^{N_g}(1 - g_i^e) \cdot \prod_{j=1}^{N_m}(1 - m_j^e) \quad (1)$$

where g_i^e defines the error rate for single or bi-qubit gate and m_j^e the error rate or measurement, respectively. ESP is based on two assumptions: (i) each gate and measurement successfully completes or results in a complete failure of the quantum algorithm, and (ii) the failures are stochastically independent [11].

3. Approach

In this section, we discuss our proposed approach for modeling and analyzing hybrid quantum software architectures.

3.1. Modeling Hybrid Quantum Software

To model hybrid quantum software architectures, we extend the PCM. The reason for choosing PCM is that it is (i) mature and (ii) provides an expressive modeling language to describe component-based software architectures. In this section, we briefly sketch the extension of PCM to a qADL w.r.t. the base models of PCM, namely the repository, system, resource and allocation model.

Our approach assumes the availability of centralized platforms where quantum algorithms and their implementations are collected, as Salm et al. [5] described them. Such a platform is represented in PCM by the repository model. Recall from section 2.1 that the repository model contains all components and interfaces that connects components. Because a quantum algorithm (such as Shor) can have several implementations, we represent each implementation as a single component that provides the same interface. Thus, on the metamodel-level of PCM, a new component class is introduced that represents a quantum component. The attributes of a quantum component specify besides meta-information about the algorithm (e.g., name and description) the required SDK based on [5] (we defer the discussion for this decision to section 3.2). Each implementation of a quantum algorithm shares the same providing interface which specifies the input and output parameters. Note that we deliberately do not include further quantum algorithm characteristics in the interface definition. Thus, we abstract away details that are not relevant at architectural level, e.g., executing quantum algorithms in a workflow-based manner [4].

In the system model of PCM, quantum components can be instantiated and assembled (w.r.t. the providing and requiring interfaces) in the same way as classical components.

In order to describe the available hardware to which components can be deployed, PCM provides a resource environment model. We extend the resource environment model by enabling the modeling of quantum computers. Each quantum computer has the attributes: Number of qubits, maximum depth (estimated by decoherence time and maximum gate time [5]), SDK and other descriptive attributes (we also defer the discussion here to section 3.2).

Finally, the allocation model describes the deployment of classic components on classic hardware and quantum components on quantum hardware. At this stage, the *Object Constrained Language* (OCL) [13] can be used to enforce at metamodel level that no classic components are deployed on quantum hardware and vice versa. Figure 2 shows a simplified example of our proposed extensions to the four PCM models.

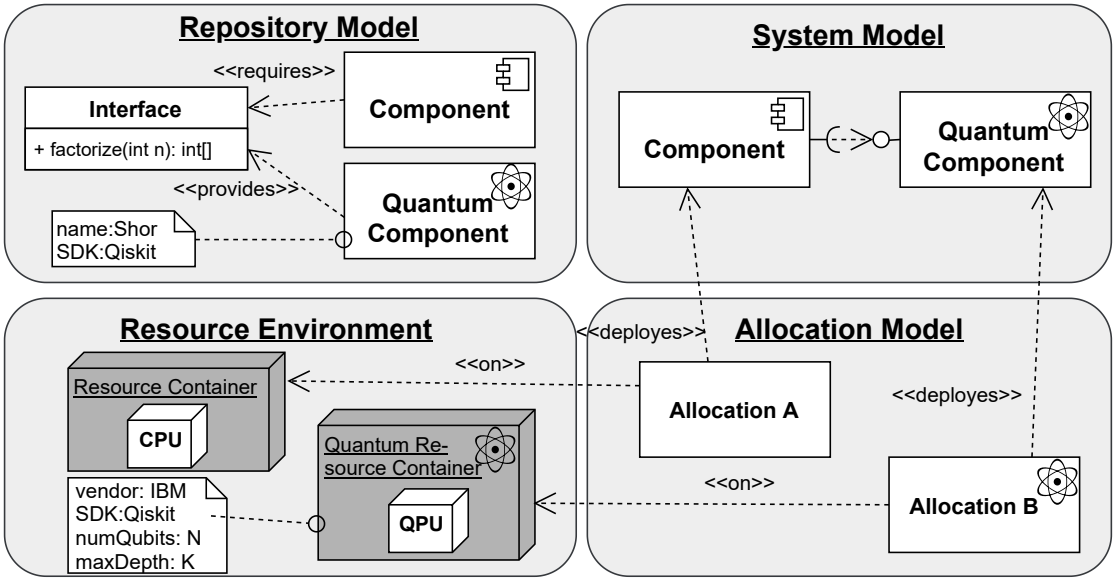

Figure 2: Example model of our proposed qADL (simplified)

Based on the extension of PCM to a qADL, we are able to model hybrid quantum software architectures and analyze them in terms of quality attributes.

3.2. Analyzing Reliability Properties

With the proposed qADL, we can model and anlyze hybrid software architectures. Because the qADL is based on PCM, we propose to extend one of the reliability analysis tools provided by PCM, namely *PCM-Rel*. Recall from section 2.1 that *PCM-Rel* transforms a PCM model instance to a DTMC based on failure annotations. In *PCM-Rel*, components can be annotated with software failures, i.e., the failure probability. Similarly, we extend *PCM-Rel* such that quantum components can be annotated with software failures and corresponding failure probabilities. In *PCM-Rel*, failure probabilities are traditionally determined by domain experts. We propose to estimate the failure probability of quantum components with the ESP metric from section 2.2. However, the reliability of a quantum component depends on the quantum computer used for execution.

Recall that our proposed qADL allows the modeling of quantum components and quantum resource container (i.e., quantum computer) with specific attributes (name, description, required SDK, number of qubits and maximum depth) based on the work of Salm et al. [5]. They describe

a method to automatically select quantum computers for a given implementation of a quantum algorithm by checking whether the implementation is executable on a given quantum computer w.r.t. the aforementioned attributes. To check a particular combination, they use the vendor-specific SDK to transpile the implementation to the particular quantum computer. Based on the transpiled hardware-specific quantum circuit, the width and depth of the implementation can be determined and compared against the capabilities of the hardware.

For our approach, we reuse this method to obtain the transpiled circuit for a particular quantum computer. Based on the transpiled circuit, we estimate the failure probability of the implementation on the particular quantum computer FP_Q by applying ESP, i.e., $FP_Q = 1 - ESP$. For a hybrid quantum software architecture modeled with our qADL, FP_Q is calculated for each quantum algorithm implementation and adjusted in the model. The adjusted model in turn is forwarded to *PCM-Rel* to predict the overall reliability of the system. Please note, that analyzing implementations of quantum algorithms is not contradicting our goal to support quantum software engineers during design phase, as we assumed the availability of repositories of quantum algorithms and their implementations (described in section 3.1). We expect quantum software engineers to embed these implementations into hybrid quantum software architectures to build applications.

3.3. Exploring the Design Space

Designing software systems includes numerous design decisions. Depending on the complexity of the software architecture, the set of possible architectural candidates (i.e., the design space) is very large. A software engineer, however, must explore the design space in order to identify an architectural candidate that satisfies the non-functional requirements. In hybrid quantum software architectures, the design space is enriched by quantum-specific design decisions, namely the choice of an implementation for a particular quantum algorithm and the choices of a quantum computer on which an implementation can be deployed and executed [5]. Hence, the design space grows and makes it even more difficult for a software engineer to find the best architectural candidate.

Therefore, we advocate the use of *PerOpteryx*. Recall from section 2.1 that *PerOpteryx* provides a method to automatically explore the design space of PCM models. The core of *PerOpteryx* forms a degree of freedom metamodel that encodes several design decisions, e.g., deployment options of a component. We propose to extend *PerOpteryx* by the inclusion of quantum-specific design decisions. Thus, a hybrid quantum software architecture is first modeled using our qADL and design decisions (classical and quantum-specific) are encoded by a degree of freedom model instance. Finally, *PerOpteryx* is applied for an automated design space exploration. Figure 3 depicts this idea.

Internally, *PerOpteryx* implements an evolutionary algorithm for design space exploration. Candidates are generated by combining selections of design decisions based on evolutionary strategies. Each candidate is evaluated by applying the analysis tools provided by the Palladio framework. The outcome of an analysis run of *PerOpteryx* is the set of Pareto-optimal candidates, from which a software engineer can select the candidate that balances the quality attributes best. In our case, we apply our reliability prediction approach from section 3.2 to obtain reliability estimations of several hybrid quantum software architecture model candidates.

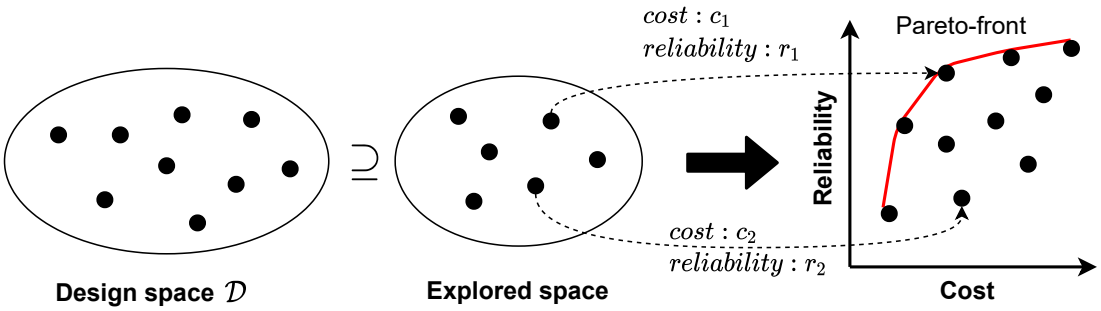

Figure 3: Exploring a design space using *PerOpetryx*

PCM also provides a cost analysis tool where resource containers can be annotated with additional costs. These cost estimates can be enriched with our reliability predictions such that the result of a *PerOpteryx* run is a Pareto-optimal set of hybrid quantum software architecture candidates that balances cost and reliability of an architecture. Thus, a software engineer is supported in making trade-off decisions for reliability and costs.

4. Related Work

Proposed modeling approaches can be characterized by the software design level they target. On the lowest design level – often called detailed design level – the modularisation of quantum circuits is addressed. Ali and Yue [14] raised several questions on modeling quantum circuits and proposed an extension of UML to model quantum circuits. In particular they used state machine diagrams to document the state (data) encoded by entangled qubits after each gate operation. Pérez-Castillo et al. [15] propose to extend UML to specify quantum circuits by activity diagrams. Exman and Shmilovich [16] show that the theory of linear software models can be used to modularise quantum programs (circuits), the same way it is used for classical programs.

On a higher design level Pérez-Delgado and Perez-Gonzalez [17] used UML class and sequence diagrams to distinguish between quantum and non-quantum structures of a hybrid quantum application along with their interactions. Gemeinhardt et al. [18] discuss research questions raised by applying model-driven engineering (MDE) to hybrid quantum applications. One of the questions targets deployment methods for cloud-based quantum computing. While Salm et al. [5] developed the *NISQ Analyzer* tool to support selecting suitable quantum processors for deploying a quantum application, we propose to adapt the Palladio framework to model and simulate software architectures [6] and the *PerOpteryx* optimizer [12] to achieve a holistic optimization of a hybrid quantum application — considering classical and quantum software components. Hence, we target the architectural design level.

For modeling quantum errors, Weber et al. [19] use *Ishikawa* diagrams to model cause-effect relationships of quantum noise. Nishio et al. [9] introduced a compiler, optimizing the success of a quantum algorithm by respecting the individual error rates of qubits, see section 2.2.

5. Conclusion and Future Work

In this paper, we proposed a model-based approach for modeling and analyzing hybrid quantum software architectures. First we presented an qADL by extending the PCM. Moreover, we extended the reliability analysis tool of PCM by estimating the reliability of quantum software components using the ESP metric. Finally, we sketched how to integrate the approach in the design space exploration tool *PerOpteryx* to support the decision making process of a software engineer. Although, we discussed only reliability analysis, the proposed qADL can be extended to analyze other quality attributes, e.g., performance.

In future work, we plan to implement and refine our approach. Also, we plan to validate our approach by comparing the design-time predictions with runtime results of sample applications.

Acknowledgments

This work is part of the SEQUOIA project funded by the Ministry of Economic Affairs Baden-Württemberg, Germany.

References

[1] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM review 41 (1999) 303–332.

[2] F. Bova, A. Goldfarb, R. G. Melko, Commercial applications of quantum computing, EPJ Quantum Technol. 8 (2021).

[3] M. Salm, J. Barzen, F. Leymann, B. Weder, About a criterion of successfully executing a circuit in the nisq era: what $wd << 1/\epsilon_{eff}$ really means, in: Proceedings of the 1st ACM International Workshop on Architectures and Paradigms for Engineering Quantum Software, APEQS 2020, ACM, 2020, pp. 10–13.

[4] F. Leymann, J. Barzen, Hybrid quantum applications need two orchestrations in superposition: A software architecture perspective, 2021. arXiv:2103.04320.

[5] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, K. Wild, The nisq analyzer: Automating the selection of quantum computers for quantum algorithms, in: Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing, Springer, 2020, pp. 66–85.

[6] R. Reussner, S. Becker, J. Happe, R. Heinrich, A. Koziolek, Modeling and simulating software architectures: The Palladio approach, MIT Press, 2016.

[7] J. Zhao, Quantum software engineering: Landscapes and horizons, 2020. arXiv:2007.07047.

[8] F. Brosch, H. Koziolek, B. Buhnova, R. Reussner, Architecture-based reliability prediction with the palladio component model, IEEE Transactions on Software Engineering 38 (2011) 1319–1339.

[9] S. Nishio, Y. Pan, T. Satoh, H. Amano, R. V. Meter, Extracting success from ibm's 20-qubit machines using error-aware compilation, ACM Journal on Emerging Technologies in Computing Systems (JETC) 16 (2020) 1–25.

[10] S. S. Tannu, M. Qureshi, Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes, in: Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO '52, ACM, 2019, p. 253–265.

[11] J. Liu, H. Zhou, Reliability modeling of nisq-era quantum computers, in: 2020 IEEE International Symposium on Workload Characterization (IISWC), IEEE, 2020, pp. 94–105.

[12] A. Martens, H. Koziolek, S. Becker, R. Reussner, Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms, in: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering, ACM, 2010, pp. 105–116.

[13] J. Cabot, M. Gogolla, Object Constraint Language (OCL): A Definitive Guide, Springer Berlin Heidelberg, 2012, pp. 58–90.

[14] S. Ali, T. Yue, Modeling quantum programs: Challenges, initial results, and research directions, in: Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software, APEQS 2020, ACM, 2020, p. 14–21.

[15] R. Pérez-Castillo, L. Jiménez-Navajas, M. Piattini, Modelling quantum circuits with uml, in: Proceedings of the IEEE/ACM 43nd International Conference on Software Engineering Workshops, ICSEW'21, ACM, 2021, p. to appear. arXiv:2103.16169.

[16] I. Exman, A. T. Shmilovich, Quantum software models: The density matrix for classical and quantum software systems design, in: Proceedings of the IEEE/ACM 43nd International Conference on Software Engineering Workshops, ICSEW'21, ACM, 2021, p. to appear. arXiv:2103.13755.

[17] C. A. Pérez-Delgado, H. G. Perez-Gonzalez, Towards a quantum software modeling language, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW'20, ACM, 2020, p. 442–444.

[18] F. Gemeinhardt, A. Garmendia, M. Wimmer, Towards model-driven quantum software engineering, in: Proceedings of the IEEE/ACM 43nd International Conference on Software Engineering Workshops, ICSEW'21, ACM, 2021, p. to appear.

[19] T. Weber, M. Riebisch, K. Borras, K. Jansen, D. Krucker, Modelling for quantum error mitigation, in: 2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C), IEEE, 2021, pp. 102–105.