Operations Patterns for Hybrid Quantum Applications

Martin Beisel[a], Johanna Barzen[b], Frank Leymann[c] and Benjamin Weder[d]
Institute of Architecture of Application Systems, University of Stuttgart, Germany
{*lastname*}*@iaas.uni-stuttgart.de*

Keywords: Quantum Computing, Pattern Languages, Hybrid Quantum Applications, Operations.

Abstract: With the rapidly improving capabilities of today's quantum devices, the development of high-quality quantum applications, that can be utilized to solve practically relevant problems, becomes necessary. Quantum software engineering is an emerging paradigm aiming to improve the quality, reusability, and maintainability of quantum applications. However, while various concepts have been presented for developing and operating quantum applications, there is a lack of structured documentation supporting developers and operations personnel in applying the best practices. To facilitate the development of quantum applications, a pattern language for the quantum computing domain has been introduced. It documents proven solutions to commonly recurring problems during the quantum software development lifecycle in an abstract manner. However, it does not contain patterns for the operations of hybrid quantum applications. To bridge this gap, in this work, we introduce novel patterns focusing on packaging, testing, executing, and observing hybrid quantum applications.

1 INTRODUCTION

Quantum software engineering is an emerging paradigm in research and industry (Piattini et al., 2021; Zhao, 2020). Its goal is to improve the quality of quantum applications and to promote principles, such as reusability, maintainability, or separation of concerns (Weder et al., 2022). Thereby, it comprises concepts for the (i) development of quantum applications, as well as their (ii) operation and adaptation based on changing requirements or user feedback.

In classical software engineering, an established approach for documenting solutions to recurring problems in a structured manner is patterns (Alexander et al., 1977). Patterns ease the understanding of problems and the factors that make them difficult to solve. To overcome these problems, patterns abstractly describe proven solutions, enabling users to transfer them to their use case at hand. Multiple patterns of the same domain can be combined in a so-called pattern language (Alexander et al., 1977).

To support quantum software engineers in developing quantum applications, the quantum computing pattern language has been introduced by Leymann (Leymann, 2019). It contains various patterns tackling different problem areas, e.g., the encoding of classical data for quantum devices (Weigold et al., 2021a). While there are already patterns providing design concepts for the development of quantum applications, there are no patterns documenting best practices for successfully operating and managing them.

To bridge this gap, we extend the quantum computing pattern language by introducing five novel patterns for the operation and management of quantum applications. These patterns cover various phases of the operations lifecycle of quantum applications. This includes proven solution strategies for: (i) The holistic testing of quantum applications and (ii) their self-contained packaging including all relevant software artifacts. Furthermore, (iii) the selection of a suitable quantum device to successfully execute a given quantum circuit. Another pattern shows (iv) how to ease the execution of quantum circuits across various heterogeneous quantum cloud offerings. Finally, (v) concepts for monitoring and analyzing quantum applications in heterogeneous environments are documented.

The remainder of the paper is structured as follows: In Section 2, fundamentals and the pattern format used for authoring the patterns are presented. Section 3 gives an overview of the quantum computing pattern language and then introduces our operations patterns in detail. Section 4 discusses the presented operations patterns. Finally, Section 5 presents related work, and Section 6 concludes the paper.

[a] https://orcid.org/0000-0003-2617-751X
[b] https://orcid.org/0000-0001-8397-7973
[c] https://orcid.org/0000-0002-9123-259X
[d] https://orcid.org/0000-0002-6761-6243

Beisel, M., Barzen, J., Leymann, F. and Weder, B.
Operations Patterns for Hybrid Quantum Applications.
DOI: 10.5220/0013246900003950
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 15th International Conference on Cloud Computing and Services Science (CLOSER 2025)*, pages 26-36
ISBN: 978-989-758-747-4; ISSN: 2184-5042
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

2 FUNDAMENTALS

In this section, the fundamentals of hybrid quantum applications as well as the utilized pattern format and the employed pattern authoring process are discussed.

2.1 Hybrid Quantum Applications

While quantum devices promise computational advantages for various problems, e.g., in machine learning (DeBenedictis, 2018) or chemistry (Cao et al., 2018), they are unsuitable for many traditional tasks, e.g., visualizing data (Preskill, 2018). Thus, quantum devices are typically used in hybrid quantum applications to solve specific problems efficiently (Leymann and Barzen, 2020). These hybrid applications comprise classical programs as well as quantum circuits that are executed on a quantum device. To execute a quantum circuit, it must be sent to a quantum cloud offering that provides access to quantum devices via an API. However, due to the heterogeneity of quantum cloud offerings and quantum devices, e.g., in the supported circuit format, a quantum circuit can not be executed on arbitrary quantum devices.

2.2 Pattern Format

Patterns are well-structured documents describing solutions to commonly recurring problems (Alexander et al., 1977). Each pattern is identified by its *name* and a mnemonic *icon*. To clarify which problem is tackled by a pattern, each pattern comprises a brief *problem statement*. The problem statement is followed by a description of the *context* in which the problem occurs and the *forces* that complicate solving it. The *solution* section of a pattern presents a strategy for solving the previously described problem and a corresponding *solution sketch*. Subsequently, the *results* paragraph discusses the consequences of applying the solution. In the *related patterns* section, connections to other patterns within the same pattern language are discussed. Finally, real-world use cases and implementations of the pattern are showcased in the *known uses* section.

The patterns presented in this work were systematically identified using the pattern authoring method introduced by Fehling et al. (2014a). In the *pattern identification* phase, we analyzed state-of-the-art approaches in the literature and best practices promoted by different quantum cloud providers and quantum SDKs. All relevant information about the operation of quantum applications is then identified by interpreting the collected data. Finally, the pattern format is defined to fit the domain, and the patterns are written and iteratively refined in the *pattern authoring* phase.

3 OPERATIONS PATTERNS

In the following, we first give an overview of the quantum computing pattern language initiated by Leymann (2019). Subsequently, we document five novel patterns covering different aspects of the operation and management of hybrid quantum applications.

3.1 Quantum Computing Pattern Language Overview

Figure 1 provides an overview of the quantum computing pattern language. It comprises the existing patterns for designing and implementing hybrid quantum applications, as well as the newly introduced patterns for quantum application operation and management.

The patterns are categorized based on the aspects of the quantum application lifecycle they address (Weder et al., 2022): The *program flow* patterns (Weigold et al., 2021b) show how computations can be split between quantum and classical hardware. To improve the performance of quantum algorithms, the *warm-starting* category (Truger et al., 2024) comprises patterns describing different techniques for, e.g., approximating an initial solution or pre-computing parameters for optimization problems. The *data encodings* patterns (Weigold et al., 2021a) summarize various state preparation routines to encode classical data into a quantum circuit. The limited capabilities of today's quantum devices hinder the successful execution of large quantum circuits (Preskill, 2018). To reduce the size of quantum circuits, *circuit cutting* can be applied, and different techniques are described in the eponymous category (Bechtold et al., 2023). The *quantum states* patterns (Leymann, 2019) document how to create fundamental quantum states, utilized as a basis for various quantum algorithms. Another approach to reduce noise on today's quantum devices is *error handling*, e.g., by mitigating results based on the approximated error model of the used quantum device (Beisel et al., 2022). Quantum circuits can be executed in different ways, e.g., using queues or prioritized access, and the corresponding concepts are summarized by the patterns in the *execution* category (Georg et al., 2023). The *unitary transformation* patterns document typical subroutines that are used in different quantum algorithms. Complementary to the patterns for operating quantum applications introduced in this paper, the *development* patterns (Bühler et al., 2023) describe best practices for developing high-quality quantum applications. Finally, the *measurement* patterns (Weigold et al., 2021a) summarize approaches for extracting classical data from quantum devices.

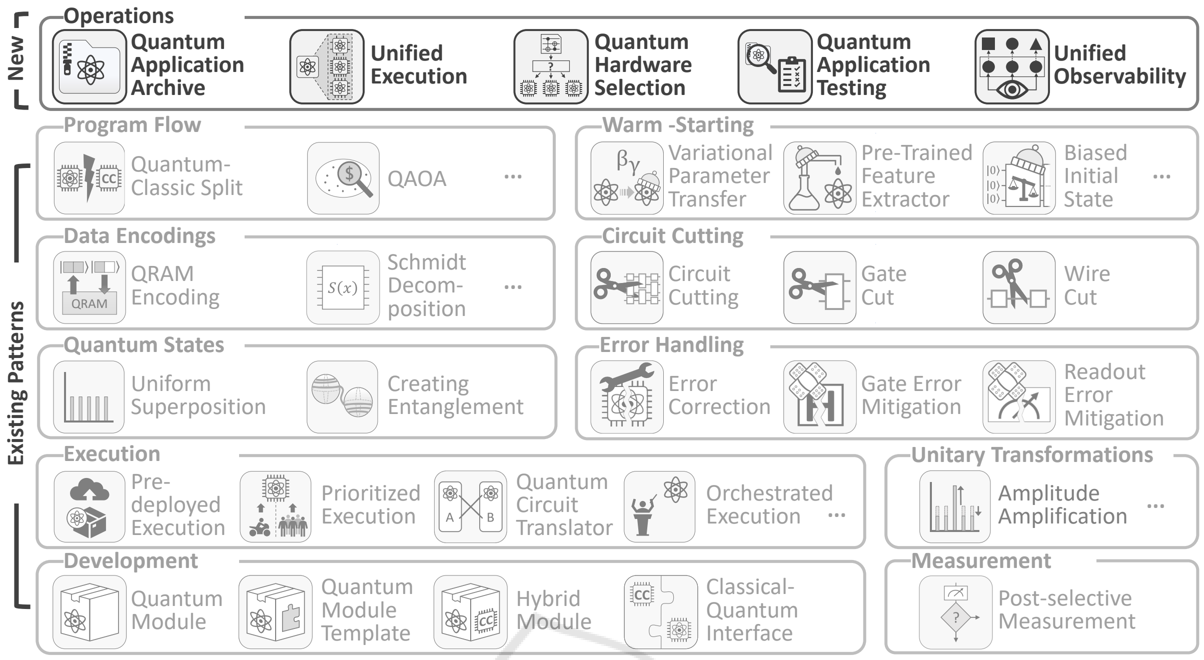

Figure 1: Overview of the quantum computing pattern language with some existing (light gray) and new patterns (dark gray).

In this work, we introduce patterns documenting abstract solutions for operating and managing hybrid quantum applications. The newly added *Operations* category in Figure 1 comprises five novel patterns: The QUANTUM APPLICATION ARCHIVE pattern focuses on the self-contained packaging of hybrid quantum applications including all required artifacts, such as quantum circuits and classical programs. To cope with the plethora of heterogeneous quantum devices, the QUANTUM HARDWARE SELECTION pattern shows how to select a suitable quantum device for the execution of a given quantum circuit. Thereby, characteristics of the quantum circuit, as well as the quantum devices and the corresponding quantum cloud offerings are taken into account. The execution across various heterogeneous quantum cloud offerings is covered by the UNIFIED EXECUTION pattern, which automatically translates required input and output formats. Thus, users can utilize their preferred programming language or SDK to implement quantum circuits and still use any desired quantum cloud offering without additional effort. To verify the correct functionality of hybrid quantum applications, the QUANTUM APPLICATION TESTING pattern describes a holistic testing strategy including all relevant artifacts. Finally, the UNIFIED OBSERVABILITY pattern focuses on the monitoring and analysis of hybrid quantum applications. It describes concepts for ensuring reproducibility and understandability while abstracting from the heterogeneity of quantum cloud offerings.

3.2 Quantum Application Archive

Problem: How to store, version, and distribute the various heterogeneous artifacts of a hybrid quantum application?

Context: Hybrid quantum applications comprise a wide variety of artifacts, e.g., classical programs, quantum circuits, deployment models, and specifications of the control and data flow (Weder et al., 2022). The execution of hybrid quantum applications in a target environment requires the availability of these artifacts. For a stable execution in production environments, the application must be versioned and all dependencies must be fixed (Altmanninger et al., 2009).
Forces: Software in the quantum computing domain is frequently updated (Vietz et al., 2021; Weder et al., 2021b). This often leads to incompatibility when upgrading different parts of the application. Further, identifying all required artifacts and transferring them into a target environment is time-consuming, complex, and error-prone. Distributing and selling hybrid quantum applications, e.g., via a marketplace, typically requires them to be available as a single entity.
Solution: To enable the storage, versioning, and distribution of hybrid quantum applications, package all required artifacts in a self-contained quantum application archive as shown in Figure 2. Thereby, ensure that all artifacts have a fixed version to prevent incompatibilities through future updates. The quantum ap-

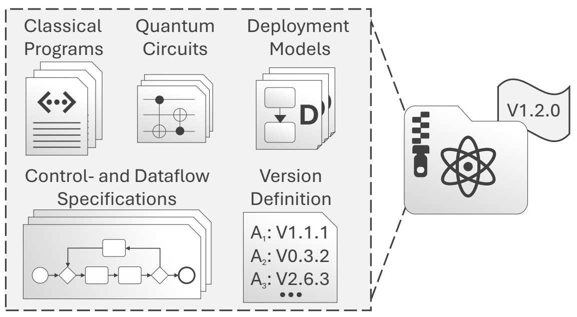

Figure 2: Solution sketch for the QUANTUM APPLICATION ARCHIVE pattern, comprising all relevant artifacts.

plication archive must comprise all artifacts required to set up the execution environment of the application and to subsequently execute it in this environment.
Result: By packaging quantum applications utilizing a quantum application archive, all classical and quantum artifacts required for execution can be shared as a single entity. Since the archive does not only contain the executable programs but also comprises all the necessary data for setting up the execution environment, it can be executed independently of the execution environment installed by the user. Due to the holistic versioning of the artifacts of the quantum application archive, incompatibilities are prevented.
Related Patterns: Hybrid quantum algorithms within a quantum application archive are commonly realized by the HYBRID MODULE pattern (Bühler et al., 2023). To deploy and execute the packaged quantum application, the PRE-DEPLOYED EXECUTION pattern (Georg et al., 2023) can be utilized. The QUANTUM APPLICATION TESTING pattern should be applied when packaging a hybrid quantum application to verify the correct behavior of all contained artifacts.
Known Uses: Weder et al. (2021b) introduce a concept for the self-contained packaging of hybrid quantum applications. Different development lifecycles for hybrid quantum applications have been presented that include the packaging of the application as an essential phase (Gheorghe-Pop et al., 2020; Weder et al., 2022). Leymann et al. (2019) describe a platform for distributing quantum applications, which enables users to package and sell their quantum applications.

3.3 Unified Execution

Problem: How to execute a quantum circuit independently of the heterogeneous quantum cloud offerings and their supported quantum circuit formats?
Context: Quantum circuits should be executed using a suitable quantum cloud offering. The circuits might be implemented utilizing different SDKs, such as Qiskit and Braket, quantum programming languages, such as Q#, or quantum assembler, such as OpenQASM (Leymann et al., 2020; Vietz et al., 2021).
Forces: When an SDK is used to implement quantum circuits, they can only be executed utilizing a quantum cloud offering supported by the respective SDK. If, on the other hand, a quantum programming language or a quantum assembler is used, the quantum cloud offering for the execution must support this technology. This often leads to a vendor lock-in, which prevents users from flexibly switching to a different quantum cloud offering that provides, e.g., cheaper access.
Solution: Unify the execution of quantum circuits by utilizing a middleware providing a single, unified interface for accessing different quantum cloud offerings. Figure 3 gives an overview of the conceptual structure of the unification middleware. This middleware uses a set of translators that automatically translate the given quantum circuit for the target quantum device if the format of the circuit is not natively supported. Since accessing quantum devices requires users to authenticate themselves, an access token is required for each quantum cloud offering. They can either (i) be provided with each execution request or (ii) the middleware can store the tokens of each user. Alternatively, (iii) the middleware provides access to all quantum devices via a separate pricing model, facilitating the execution for users as they do not require an access token for each quantum cloud offering.

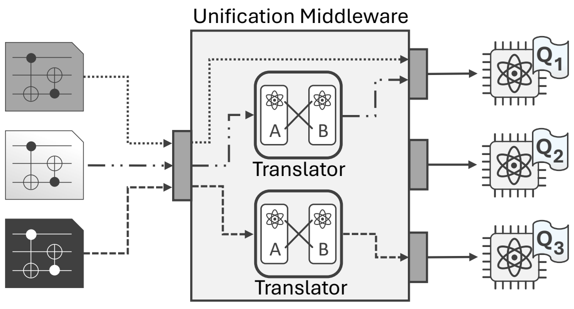

Figure 3: Solution sketch for the UNIFIED EXECUTION pattern, automatically translating quantum circuits.

Result: The quantum circuit can be executed via a single interface independently of the circuit format and quantum cloud offering. Thus, vendor lock-ins can be avoided and circuits can be executed using different quantum cloud offerings without any additional effort. However, the unification middleware does not automatically select a suitable quantum device but only supports the execution on the target device.
Related Patterns: The QUANTUM CIRCUIT TRANSLATOR pattern (Bühler et al., 2023) is utilized by the unification middleware to translate quantum circuits that use data formats incompatible with the target quantum device. To select a suitable quantum de-

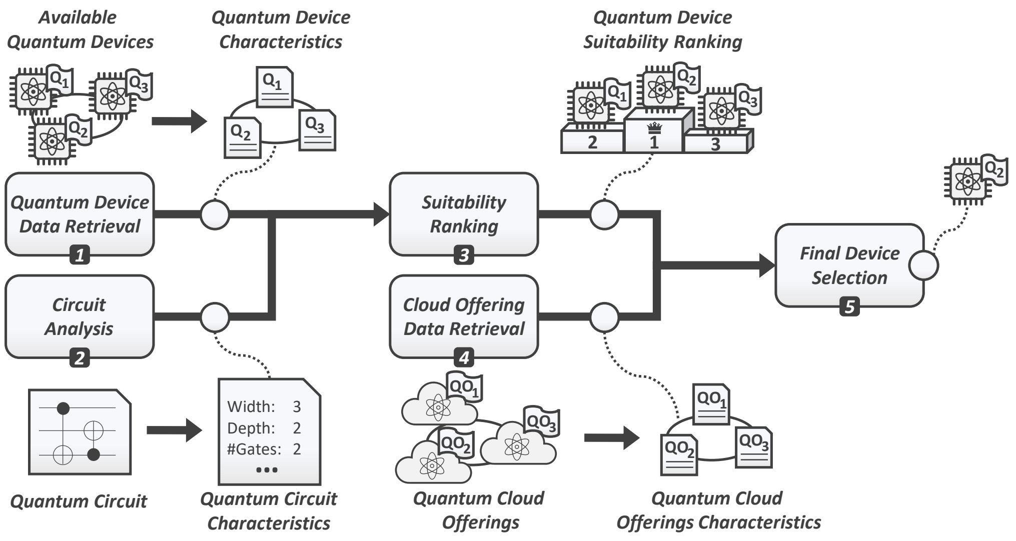
Available Quantum Devices
Quantum Device Characteristics
Quantum Device Suitability Ranking
Q1
Q2
Q3
2
1
3
Quantum Device Data Retrieval
1
Circuit Analysis
2
Suitability Ranking
3
Cloud Offering Data Retrieval
4
Final Device Selection
5
Width: 3
Depth: 2
#Gates: 2
QO1
QO2
QO3
Quantum Circuit
Quantum Circuit Characteristics
Quantum Cloud Offerings
Quantum Cloud Offerings Characteristics

Figure 4: Solution sketch for the QUANTUM HARDWARE SELECTION pattern, showing the phases to select a quantum device.

vice for the quantum circuit at hand, the QUANTUM HARDWARE SELECTION pattern can be used in combination with the UNIFIED EXECUTION pattern. The UNIFIED OBSERVABILITY pattern enables monitoring and analyzing the execution of quantum circuits when using heterogeneous quantum cloud offerings.
Known Uses: Giortamis et al. (2024) introduce *Qonductor*, a tool for executing quantum applications via a hardware-agnostic API. Salm et al. (2020) present a concept as well as a corresponding tool for automatically translating circuits between different circuit formats and executing them via different quantum cloud offerings. Weder et al. (2024) introduce a unification middleware for quantum cloud offerings, which also enables additional features, such as circuit cutting.

3.4 Quantum Hardware Selection

Problem: How to automatically select a suitable quantum device to execute a given quantum circuit?

Context: Quantum circuits can either be executed on a quantum device or a classical computer simulating the computation. However, it is impossible to simulate larger quantum circuits using classical hardware (Zhou et al., 2020). Therefore, a suitable quantum device for the execution must be selected.
Forces: Quantum devices are provided by different vendors, e.g., IBM, IonQ, and Rigetti (Leymann et al., 2020). These quantum devices are very heterogeneous and differ in characteristics, such as the number of qubits, their decoherence times, or the supported gate set (Weder et al., 2021a). Some of the characteristics change over time, e.g., the decoherence times when recalibrating the quantum device (Tannu and Qureshi, 2019). However, the successful execution of a given quantum circuit depends on these characteristics (Salm et al., 2020). Thus, selecting an unsuitable quantum device can lead to error-prone results. Quantum cloud offerings also differ regarding their payment models and access methods, e.g., queue-based systems or reservations of exclusive time slots.
Solution: Figure 4 gives an overview of the phases to select suitable quantum devices. First, the characteristics of the quantum devices available to the user are retrieved. This can either be done by periodically accessing an API providing data about these characteristics, e.g., a provider API or a dedicated provenance system, or by executing benchmarks that approximate device characteristics (IBM, 2024a; Tomesh et al., 2022; Weder et al., 2021a). Then, analyze the given quantum circuit so that in the next phase the suitability of the devices can be ranked based on the characteristics of the circuits and devices (Salm et al., 2020). Finally, ensure that a quantum device is selected that is available via a suitable cloud offering, e.g., a cloud offering supporting a pay-per-use model.
Result: The quantum circuit can be executed on the selected quantum device. Through proper hardware selection, the impact of errors can be reduced and other factors, such as the waiting time, can be optimized. If the format of the circuit is incompatible with the selected quantum device, it must be translated.
Related Patterns: The UNIFIED EXECUTION pattern can be used to execute the quantum circuit if the circuit format is not supported by the selected

quantum device. The PRIORITIZED EXECUTION pattern (Georg et al., 2023) enables selecting a suitable quantum device even if the respective queue is long by providing prioritized access to the quantum device. Since errors still occur when utilizing a suitable quantum device, their impact can be reduced by applying the READOUT ERROR MITIGATION and GATE ERROR MITIGATION patterns (Beisel et al., 2022).
Known Uses: Salm et al. (Salm et al., 2020) and Quetschlich et al. (Quetschlich et al., 2023) introduce tools to automatically select suitable quantum devices based on given quantum circuits. Suchara et al. (Suchara et al., 2013) present the *QuRE Toolbox*, a framework to estimate the required resources for executing quantum circuits. This information can then be used to select a suitable quantum device. Qiskit provides functionalities to filter available quantum devices based on different characteristics, e.g., the minimum number of qubits or the supported gate set (IBM, 2024b). Further, the quantum device for the execution can be selected from the remaining quantum devices based on their current queue size.

3.5 Quantum Application Testing

Problem: How to ensure the correctness of all functionalities of a hybrid quantum application?

Context: Hybrid quantum applications are realized using a plethora of different artifacts, such as quantum circuits, classical programs, deployment models, and control and data flow specifications (Weder et al., 2021b). The correctness of the functionality of all artifacts as well as their interactions must be ensured.
Forces: Quantum applications comprise heterogeneous programs, e.g., using different programming languages and data formats. The execution of quantum circuits is probabilistic and arbitrary unknown quantum states can not be copied, hence, obtaining information about a qubit without disturbing the corresponding quantum system state is impossible (Ali et al., 2021; Bužek and Hillery, 1996). Further, simulating the execution of larger quantum circuits is impossible due to the exponential resources required for simulating additional qubits (Zhou et al., 2020). The changing characteristics of quantum devices may lead to different results when executing the same quantum circuit at different times, even when using the same quantum devices (Tannu and Qureshi, 2019).
Solution: Utilize a holistic testing strategy comprising the following steps as depicted in Figure 5: (i) Unit tests for the classical programs. (ii) Specific tests for the quantum circuits. This includes mathematical verification of quantum circuits (Chareton et al., 2021; Wang et al., 2008), adding and evaluating assertions to ensure certain states (Huang and Martonosi, 2019; Liu et al., 2020), as well as white and black box tests for quantum circuits (Miranskyy et al., 2020). (iii) Deployment tests verifying that the application was provisioned as intended (Wurster et al., 2018). (iv) Integration tests validating the interplay of the various software artifacts (Wu et al., 2003).

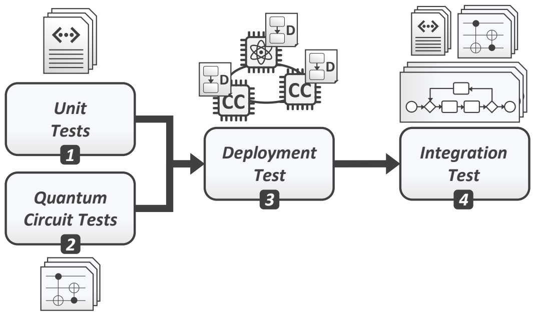

Figure 5: Solution sketch for the QUANTUM APPLICATION TESTING, documenting four testing phases.

Result: By testing all artifacts of a quantum application, as well as their interplay and execution environment, the reliability of the quantum application is significantly increased. Well-tested artifacts of hybrid applications promote their reuse for other applications (Weder et al., 2022; Zhao, 2020). To automate the testing procedure it may be integrated, e.g., into the application's continuous integration and development (CI/CD) pipeline (Romero-Álvarez et al., 2024).
Related Patterns: The correct functionality of hybrid quantum applications also depends on hardware characteristics and the current calibration of quantum devices. To select a suitable quantum device for testing the quantum application, the QUANTUM HARDWARE SELECTION pattern can be utilized. The QUANTUM APPLICATION ARCHIVE pattern can include test specifications enabling integration and deployment tests in the target environment. Due to the increased complexity when integrating quantum and classical programs, each CLASSICAL-QUANTUM INTERFACE (Bühler et al., 2023) should be tested.
Known Uses: Different lifecycles for hybrid quantum applications include a dedicated testing phase focussing on the quantum-specific as well as integration aspects (Gheorghe-Pop et al., 2020; Weder et al., 2022; Zhao, 2020). Becker et al. (2023) introduce a testing pipeline for hybrid quantum applications that includes tests for classical programs as well as quantum circuits. Romero-Álvarez et al. (2024) present a concept to integrate hybrid quantum applications into CI/CD pipelines. They enable the automated testing and deployment of hybrid quantum applications including the classical programs and quantum circuits.

3.6 Unified Observability

Problem: How to ensure reproducibility, understandability, and quality when executing hybrid quantum applications?

Context: Quantum applications comprise a multitude of classical programs and quantum circuits which are typically executed in a heterogeneous execution environment, e.g., utilizing classical and quantum cloud offerings (Beisel et al., 2024b; Leymann et al., 2020).

Forces: When executing hybrid quantum applications in heterogeneous execution environments, collecting all necessary data is difficult: The data must be gathered using the different APIs or SDKs of the utilized quantum and classical cloud offerings, which often change, e.g., when new features are released. Furthermore, the offerings might not provide all the required data (Weder et al., 2021a). Finally, some of the data also changes over time, e.g., the qubit decoherence times or the error rates (Tannu and Qureshi, 2019).

Analyzing the data is complicated by different data formats and abstraction levels of the provided data (Beisel et al., 2024b). Moreover, hybrid quantum applications are typically developed and operated by interdisciplinary teams with various backgrounds, e.g., physics, mathematics, and software engineering, requiring different information (Weder et al., 2022).

Solution: Figure 6 shows the phases required to achieve unified observability. Data about the execution of the quantum application and the used quantum and classical resources must be collected continuously. Persistently store these data using a provenance system that automatically unifies data using transformation methods. For example, quantum cloud offerings use both the *fidelity* and *error rate* metrics to describe the quality of their gate operations, where $\textit{Error Rate} = 1 - \textit{Fidelity}$. Use benchmarks to retrieve data that is not provided by the cloud offerings but required by the user (Tomesh et al., 2022). To enable user-group-specific monitoring and analysis provide suitable abstractions, e.g., by aggregating data or hiding unnecessary information (Beisel et al., 2024b).

Result: The provenance system stores all relevant data produced by the quantum applications, enabling their monitoring and analysis, e.g., to identify errors and optimize the application. Data abstractions facilitate understanding the application and its execution environment, particularly by visualizing crucial data.

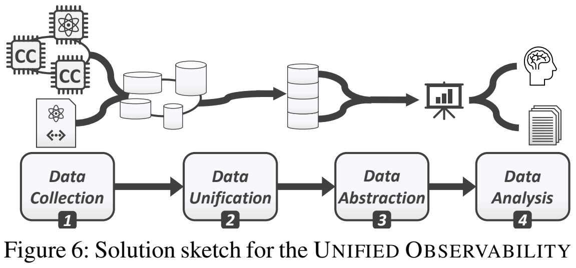

Figure 6: Solution sketch for the Unified Observability pattern, documenting the four observability phases.

Related Patterns: The Unified Execution pattern can be combined with the Unified Observability pattern to monitor and analyze the execution of quantum applications using heterogeneous hardware. The stored data about quantum devices can be used as a basis for the Quantum Hardware Selection, Gate Error Mitigation, and Readout Error Mitigation patterns (Beisel et al., 2022).

Known Uses: Beisel et al. (2024b) introduce a concept to unify the observability of hybrid quantum applications realized using workflows within heterogeneous multi-cloud environments. Weder et al. (2021a) present a provenance system for quantum computing automatically gathering required provenance data, which can be used for monitoring and analysis.

4 DISCUSSION

The validity of patterns in the software engineering domain can be confirmed by identifying several real-world occurences (Gamma et al., 1995; Schmidt et al., 1996). For each documented operations pattern for hybrid quantum applications, a set of known real-world uses is summarized in the known uses section.

Quantum software engineering is currently still a new discipline (Piattini et al., 2021; Zhao, 2020): Quantum applications are often developed research-driven and in an ad-hoc manner to realize experiments or evaluate the suitability of quantum applications to solve a certain problem (Ali et al., 2022). However, with the increasing capabilities of quantum devices, the development of high-quality hybrid quantum applications is of vital importance (Weder et al., 2022). With the usage of hybrid quantum applications in production environments, we expect a focus shift to the operational aspects of quantum software engineering and the corresponding tools provided by different research institutions and companies. Hence, additional known uses for the presented patterns can be added.

The application of the Unified Execution and Unified Observability pattern becomes increasingly more difficult the more heterogeneous the data and interfaces are. A well-established method for reducing heterogeneity is the introduction of standards. These standards are typically developed by the major stakeholders from industry as well as experts from academia. As quantum software engineering is still in its infancy, there currently is a lack of standardization. The first steps for developing dedicated standards for quantum software engineering are being taken, e.g., the introduction of OpenQASM 3.0 (Cross et al., 2022), a language for specifying quantum circuits. However, additional standards are required to improve the portability and interoperability of quantum applications and to streamline their development and operation. For example, standards for defining the capabilities of a quantum device and for storing the measurement results of a quantum computation. When operating quantum applications, often existing standards from classical software engineering can be used with no or only minor adaptations (Beisel et al., 2024a): (i) BPMN (OMG, 2011) for orchestrating the control- and dataflow of the different functionalities using workflow technologies, (ii) TOSCA (OASIS, 2013) for automating the deployment of the required programs, (iii) PROV (W3C, 2013) for storing provenance data about quantum applications, and (iv) OpenAPI (OpenAPI Initiative, 2024) for describing the interfaces of quantum and classical services.

5 RELATED WORK

The patterns for operating hybrid quantum applications introduced in this paper extend the existing quantum computing pattern language (Beisel et al., 2022; Georg et al., 2023; Leymann, 2019; Truger et al., 2024). In addition to the quantum computing pattern language, there are other works presenting and utilizing patterns within the quantum domain that do not follow the pattern structure introduced by Alexander et al. (1977). Khan et al. (2023) conduct a systematic literature review aiming to identify architecture design patterns in quantum applications. Baczyk et al. (2024) present a variety of architectural software patterns in the quantum software engineering domain. However, their patterns only focus on the development and not the operations of quantum applications. Gilliam et al. (2019) and Perdrix (2007) present various patterns for building and improving quantum circuits. Huang and Martonosi (2019) introduce a strategy for using quantum programming patterns to prevent bugs in quantum circuits. Perez-Castillo et al. (2024) analyze how different patterns from the quantum computing pattern language can be utilized to build quantum circuits using Qiskit and OpenQASM.

Some patterns presented in this work are composite patterns employing patterns that were introduced in other domains as part of their solutions: The UNIFIED EXECUTION pattern utilizes the API GATEWAY pattern (Richardson, 2018) to provide a single interface unifying the underlying quantum devices. To reduce the upper limit of required translators per language from n^2 to $2n$, a CANONICAL DATA MODEL (Hohpe and Woolf, 2004) can be used as an intermediate format for translating quantum circuits. Furthermore, the LOG DEPLOYMENTS AND CHANGES and the LOG AGGREGATION patterns (Richardson, 2018) are utilized by the UNIFIED OBSERVABILITY pattern to keep track of the execution environments of hybrid quantum applications. The HYBRID DEVELOPMENT ENVIRONMENT pattern (Fehling et al., 2014b) can be employed to use different execution environments for the development, testing, and operations of quantum applications.

To facilitate the applications of patterns, Falkenthal et al. (2017) propose the concept of solution languages. Solution languages associate so-called concrete solutions with the patterns of a pattern language. Concrete solutions are implementations of a pattern, e.g., a Java program or a quantum circuit. Thus, solution languages enable a systematic, pattern-centric collection of real-world implementations. Since concrete solutions are interconnected with the respective patterns, developers can reuse an existing implementation for their use case when utilizing a pattern. Thereby, the manual effort of implementing the abstract solution described in the pattern is reduced.

6 CONCLUSION & OUTLOOK

In this work, we motivated the need for concepts to successfully operate hybrid quantum applications and the suitability of using patterns to formalize these concepts concisely and easily digestibly. We captured existing concepts from the literature to operate and manage quantum applications and documented them as five novel patterns that extend the quantum computing patterns language. The introduced patterns are made publicly available via the *Pattern Atlas* (Leymann and Barzen, 2021), a graphical tool for authoring, managing, and visualizing patterns.

The operation and management of hybrid quantum applications is a highly active area that is gaining attention from research and industry. Thus, we plan to continue analyzing the literature and emerging tools to identify and document novel patterns in this area.

ACKNOWLEDGEMENTS

This work was partially funded by the BMWK projects *EniQmA* (01MQ22007B) and *SeQuenC* (01MQ22009B).

REFERENCES

Alexander, C., Ishikawa, S., and Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

Ali, S., Arcaini, P., Wang, X., and Yue, T. (2021). Assessing the Effectiveness of Input and Output Coverage Criteria for Testing Quantum Programs. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 13–23.

Ali, S., Yue, T., and Abreu, R. (2022). When Software Engineering Meets Quantum Computing. *Communications of the ACM*, 65(4):84–88.

Altmanninger, K., Seidl, M., and Wimmer, M. (2009). A survey on model versioning approaches. *International Journal of Web Information Systems*, 5(3):271–304.

Baczyk, B., Pérez-Castillo, R., and Piattini, M. (2024). Towards a Framework of Architectural Patterns for Quantum Software Engineering. In *Proceedings of the 4th International Workshop on Quantum Software Engineering and Technology (Q-SET)*, pages 41–48.

Bechtold, M., Barzen, J., Beisel, M., Leymann, F., and Weder, B. (2023). Patterns for Quantum Circuit Cutting. In *Proceedings of the 30th Conference on Pattern Languages of Programs (PLoP)*, pages 1–11. HILLSIDE.

Becker, C. K.-U., Gheorghe-Pop, I.-D., and Tcholtchev, N. (2023). A testing pipeline for quantum computing applications. In *Proceedings of the 2023 IEEE International Conference on Quantum Software (QSW)*, pages 54–59. IEEE.

Beisel, M., Alvarado-Valiente, J., Barzen, J., Leymann, F., Romero-Álvarez, J., Stiliadou, L., and Weder, B. (2024a). Utilizing a Standards-Based Toolchain to Model and Execute Quantum Workflows. In *Web Engineering – ICWE 2024*, pages 401–405. Springer.

Beisel, M., Barzen, J., Leymann, F., Stiliadou, L., and Weder, B. (2024b). Observability for Quantum Workflows in Heterogeneous Multi-cloud Environments. In *Advanced Information Systems Engineering – CAiSE 2024*, pages 612–627. Springer.

Beisel, M., Barzen, J., Leymann, F., Truger, F., Weder, B., and Yussupov, V. (2022). Patterns for Quantum Error Handling. In *Proceedings of the 14th International Conference on Pervasive Patterns and Applications (PATTERNS)*, pages 22–30. Xpert Publishing Services (XPS).

Bühler, F., Barzen, J., Beisel, M., Georg, D., Leymann, F., and Wild, K. (2023). Patterns for Quantum Software Development. In *Proceedings of the 15th International Conference on Pervasive Patterns and Applications (PATTERNS)*, pages 30–39. Xpert Publishing Services (XPS).

Bužek, V. and Hillery, M. (1996). Quantum copying: Beyond the no-cloning theorem. *Physical Review A*, 54(3):1844.

Cao, Y., Romero, J., and Aspuru-Guzik, A. (2018). Potential of quantum computing for drug discovery. *IBM Journal of Research and Development*, 62(6):6:1–6:20.

Chareton, C., Bardin, S., Bobot, F., Perrelle, V., and Valiron, B. (2021). An Automated Deductive Verification Framework for Circuit-building Quantum Programs. In *European Symposium on Programming (ESOP)*, pages 148–177. Springer.

Cross, A., Javadi-Abhari, A., Alexander, T., Beaudrap, N. D., Bishop, L. S., Heidel, S., et al. (2022). OpenQASM 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3):1–50.

DeBenedictis, E. P. (2018). A Future with Quantum Machine Learning. *Computer*, 51(2):68–71.

Falkenthal, M., Barzen, J., Breitenbücher, U., and Leymann, F. (2017). Solution Languages: Easing Pattern Composition in Different Domains. *International Journal on Advances in Software*, 10(3):263–274.

Fehling, C., Barzen, J., Breitenbücher, U., and Leymann, F. (2014a). A Process for Pattern Identification, Authoring, and Application. In *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM.

Fehling, C., Leymann, F., Retter, R., Schupeck, W., and Arbitter, P. (2014b). *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., and Patterns, D. (1995). *Elements of reusable object-oriented software*, volume 99. Addison-Wesley Reading.

Georg, D., Barzen, J., Beisel, M., Leymann, F., Obst, J., Vietz, D., Weder, B., and Yussupov, V. (2023). Execution Patterns for Quantum Applications. In *Proceedings of the 18th International Conference on Software Technologies (ICSOFT)*, pages 258–268. SciTePress.

Gheorghe-Pop, I.-D., Tcholtchev, N., Ritter, T., and Hauswirth, M. (2020). Quantum DevOps: Towards Reliable and Applicable NISQ Quantum Computing. In *IEEE Globecom Workshops*, pages 1–6. IEEE.

Gilliam, A., Venci, C., Muralidharan, S., Dorum, V., May, E., Narasimhan, R., and Gonciulea, C. (2019). Foundational Patterns for Efficient Quantum Computing. *arXiv:1907.11513*.

Giortamis, E., Romão, F., Tornow, N., Lugovoy, D., and Bhatotia, P. (2024). Orchestrating Quantum Cloud Environments with Qonductor.

Hohpe, G. and Woolf, B. (2004). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.

Huang, Y. and Martonosi, M. (2019). Statistical Assertions for Validating Patterns and Finding Bugs in Quantum Programs. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, pages 541–553. ACM.

IBM (2024a). IBMQ Dashboard. https://quantum.ibm.com.

IBM (2024b). Qiskit Backend Filtering. https://docs.quantum.ibm.com/guides/get-qpu-information#filter-backends.

Khan, A. A., Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Mikkonen, T., and Abrahamsson, P. (2023). Software architecture for quantum computing systems —

A systematic review. *Journal of Systems and Software*, 201:111682.

Leymann, F. (2019). Towards a Pattern Language for Quantum Algorithms. In *Proceedings of the First International Workshop on Quantum Technology and Optimization Problems (QTOP)*. Springer.

Leymann, F. and Barzen, J. (2020). The bitter truth about gate-based quantum algorithms in the NISQ era. *Quantum Science and Technology*, pages 1–28.

Leymann, F. and Barzen, J. (2021). *Pattern Atlas*, pages 67–76. Springer.

Leymann, F., Barzen, J., and Falkenthal, M. (2019). Towards a Platform for Sharing Quantum Software. In *Proceedings of the 13th Advanced Summer School on Service Oriented Computing (2019)*, IBM Technical Report (RC25685), pages 70–74. IBM Research Division.

Leymann, F., Barzen, J., Falkenthal, M., Vietz, D., Weder, B., and Wild, K. (2020). Quantum in the Cloud: Application Potentials and Research Opportunities. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER)*, pages 9–24. SciTePress.

Liu, J., Byrd, G. T., and Zhou, H. (2020). Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 1017–1030. ACM.

Miranskyy, A., Zhang, L., and Doliskani, J. (2020). Is Your Quantum Program Bug-Free? In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, page 29–32. ACM.

OASIS (2013). *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS).

OMG (2011). *Business Process Model and Notation (BPMN) Version 2.0*. Object Management Group (OMG).

OpenAPI Initiative (2024). OpenAPI Specification. https://spec.openapis.org/oas/v3.1.1.html.

Perdrix, S. (2007). Quantum Patterns and Types for Entanglement and Separability. *Electronic Notes in Theoretical Computer Science*, 170:125–138.

Perez-Castillo, R., Fernández-Osuna, M., Cruz-Lemus, J. A., and Piattini, M. (2024). A preliminary study of the usage of design patterns in quantum software. In *Proceedings of the 5th ACM/IEEE International Workshop on Quantum Software Engineering*, pages 41–48.

Piattini, M., Serrano, M., Perez-Castillo, R., Petersen, G., and Hevia, J. L. (2021). Toward a Quantum Software Engineering. *IT Professional*, 23(1):62–66.

Preskill, J. (2018). Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79.

Quetschlich, N., Burgholzer, L., and Wille, R. (2023). MQT Predictor: Automatic Device Selection with Device-Specific Circuit Compilation for Quantum Computing. *ACM Transactions on Quantum Computing*.

Richardson, C. (2018). *Microservices patterns*. Simon and Schuster.

Romero-Álvarez, J., Alvarado-Valiente, J., Moguel, E., Garcia-Alonso, J., and Murillo, J. M. (2024). Enabling continuous deployment techniques for quantum services. *Software: Practice and Experience*, 54(8):1491–1515.

Salm, M., Barzen, J., Breitenbücher, U., Leymann, F., Weder, B., and Wild, K. (2020). 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 (SummerSOC)*, pages 66–85. Springer.

Schmidt, D. C., Fayad, M., and Johnson, R. E. (1996). Software patterns. *Communications of the ACM*, 39(10):37–39.

Suchara, M., Kubiatowicz, J., Faruque, A., Chong, F. T., Lai, C.-Y., and Paz, G. (2013). QuRE: The Quantum Resource Estimator Toolbox. In *IEEE 31st International Conference on Computer Design (ICCD)*, pages 419–426. IEEE.

Tannu, S. S. and Qureshi, M. K. (2019). Not All Qubits Are Created Equal: A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. In *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, page 987–999. ACM.

Tomesh, T., Gokhale, P., Omole, V., Ravi, G. S., Smith, K. N., Viszlai, J., et al. (2022). SupermarQ: A Scalable Quantum Benchmark Suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 587–603. IEEE.

Truger, F., Barzen, J., Beisel, M., Leymann, F., and Yussupov, V. (2024). Warm-Starting Patterns for Quantum Algorithms. In *Proceedings of the 16th International Conference on Pervasive Patterns and Applications (PATTERNS)*, pages 25–31. Xpert Publishing Services (XPS).

Vietz, D., Barzen, J., Leymann, F., and Wild, K. (2021). On Decision Support for Quantum Application Developers: Categorization, Comparison, and Analysis of Existing Technologies. In *Computational Science – ICCS 2021*, pages 127–141. Springer.

W3C (2013). PROV Model Primer. World Wide Web Consortium (W3C).

Wang, S.-A., Lu, C.-Y., Tsai, I.-M., and Kuo, S.-Y. (2008). An XQDD-Based Verification Method for Quantum Circuits. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 91(2):584–594.

Weder, B., Barzen, J., Beisel, M., Bühler, F., Georg, D., Leymann, F., and Stiliadou, L. (2024). Qunicorn: A middleware for the unified execution across heterogeneous quantum cloud offerings. In *Proceedings of the 6th International Workshop on Quantum Software Engineering (Q-SE)*. IEEE.

Weder, B., Barzen, J., Leymann, F., Salm, M., and Wild, K. (2021a). QProv: A provenance system for quantum computing. *IET QuantumCommunication*, 2(4):171–181.

Weder, B., Barzen, J., Leymann, F., and Vietz, D. (2022). Quantum Software Development Lifecycle. In *Quantum Software Engineering*, pages 61–83. Springer.

Weder, B., Barzen, J., Leymann, F., and Zimmermann, M. (2021b). Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective. In *Proceedings of the 18th IEEE International Conference on Web Services (ICWS)*, pages 1–13. IEEE.

Weigold, M., Barzen, J., Leymann, F., and Salm, M. (2021a). Encoding patterns for quantum algorithms. *IET QuantumCommunication*, 2(4):141–152.

Weigold, M., Barzen, J., Leymann, F., and Vietz, D. (2021b). Patterns for Hybrid Quantum Algorithms. In *Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC)*, pages 34–51. Springer.

Wu, Y., Chen, M.-H., and Offutt, J. (2003). UML-Based Integration Testing for Component-Based Software. In *Proceedings of the 2nd International Conference on COTS-Based Software Systems*, pages 251–260. Springer.

Wurster, M., Breitenbücher, U., Kopp, O., and Leymann, F. (2018). Modeling and Automated Execution of Application Deployment Tests. In *Proceedings of the IEEE 22nd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 171–180. IEEE.

Zhao, J. (2020). Quantum Software Engineering: Landscapes and Horizons. *arXiv:2007.07047*.

Zhou, Y., Stoudenmire, E. M., and Waintal, X. (2020). What Limits the Simulation of Quantum Computers? *Physical Review X*, 10(4):041038.