Execution Patterns for Quantum Applications

Daniel Georg, Johanna Barzen, Martin Beisel, Frank Leymann,
Julian Obst, Daniel Vietz, Benjamin Weder and Vladimir Yussupov
Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{*firstname.lastname*}*@iaas.uni-stuttgart.de*

Keywords: Quantum Computing, Cloud Computing, Patterns, Pattern Language, Quantum Hardware Access.

Abstract: Continuously evolving quantum service offerings vary in development and deployment requirements they impose on quantum application developers. Further, since quantum applications often require classical pre- and post-processing steps, in addition to quantum computing knowledge, expertise in cloud service models, integration, and deployment automation is needed. Thus, to reduce the required complexity and management overhead, applications often need to be tailored for quantum offerings suited for the desired execution scenario. However, clear guidelines that facilitate deciding between diverse quantum offerings are currently missing. In this work, we bridge this gap by (i) documenting five patterns that capture different execution semantics for quantum applications. Furthermore, we (ii) analyze existing quantum offerings and document their support for the captured patterns to facilitate the decision making process when implementing quantum applications.

1 INTRODUCTION

Compared to classical hardware, quantum computers promise to solve certain computational problems in various domains, such as machine learning Gabor et al. (2020) or chemistry Cao et al. (2018), faster, with higher precision, or with less energy consumption Arute et al. (2019); Barzen (2021). However, contemporary *Noisy Intermediate-Scale Quantum (NISQ)* devices have limited capabilities and are error-prone Preskill (2018). NISQ devices from vendors such as IBM, IonQ, or Rigetti are accessible as cloud offerings LaRose (2019) that vary significantly in their capabilities. For example, graphical circuit modelers only support the execution of designed quantum circuits without any further integration features Vietz et al. (2021). In contrast, so-called hybrid runtimes support deploying quantum circuits alongside desired classical code parts and executing them together Qiskit (2023a). This is particularly helpful for quantum algorithms relying on classical parts, e.g., pre- and post-processing of data.

Since most quantum algorithms are hybrid Leymann and Barzen (2020), engineering quantum applications often requires expertise in (i) implementing quantum circuits and classical programs, and (ii) integrating them Weder et al. (2020a). While many languages and formats for the implementation of quantum algorithms exist Fingerhuth et al. (2018), they are often provider-specific and require extra effort for running them on other quantum devices, e.g., reengineering or use of compatible transpilers Sivarajah et al. (2020). Implementing for specific devices poses another challenge – their limited availability leads to long waiting queues, even for simple tasks. Thus, to implement and run quantum applications, developers need to consider both, requirements of chosen NISQ devices and operational semantics of the underlying service offerings. However, the lack of clear guidelines on different execution semantics for quantum offerings can hinder this decision making process.

One well-known way to abstractly capture solutions for problems recurring in specific contexts is to use *patterns* Alexander et al. (1977). They can also be interconnected in so-called *pattern languages* to effectively allow combining related solutions and guide architectural activities. While there exists the *quantum computing pattern language* Leymann (2019), it lacks patterns capturing different execution semantics for quantum applications. In this work, we extend this pattern language with five execution patterns for running quantum applications that we capture and document following the best practices. Further, we analyze the existing landscape of quantum offerings with respect to their support for the captured execution patterns. The resulting patterns and technology categorization aim to facilitate designing, implementing, and executing quantum applications.

Georg, D., Barzen, J., Beisel, M., Leymann, F., Obst, J., Vietz, D., Weder, B. and Yussupov, V.
Execution Patterns for Quantum Applications.
DOI: 10.5220/0012057700003538
In *Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023)*, pages 258-268
ISBN: 978-989-758-665-1; ISSN: 2184-2833
Copyright © 2023 by SCITEPRESS – Science and Technology Publications, Lda. Under CC license (CC BY-NC-ND 4.0)

2 FUNDAMENTALS

This section presents the necessary background about quantum applications and pattern authoring.

2.1 Quantum Applications

Quantum devices, e.g., quantum annealing or gate-based devices can mainly be accessed via cloud services Leymann et al. (2020). The growing popularity of gate-based devices, however, shifts the current trend towards quantum offerings that support *quantum circuits* as quantum algorithm implementations, i.e., code parts that actually run on quantum devices. Moreover, due to the hybrid nature of quantum algorithms Leymann and Barzen (2020), a typical quantum application comprises classical code parts in addition to one or more quantum circuits. These classical parts are pre- and post-processing steps required by quantum circuits, e.g., for encoding input data, optimizing parameters between quantum circuit executions, or visualizing the outputs.

To execute quantum circuits and their classical pre- and post-processing steps, a quantum offering providing classical and quantum hardware or a combination of quantum and classical offerings must be chosen. Most quantum offerings provide APIs, routines in SDKs, or graphical user interfaces that simplify interacting with the quantum offering and executing quantum circuits on the underlying quantum devices. Furthermore, to balance the load of incoming quantum circuit execution requests, quantum offerings often employ queues for fair utilization of the available quantum devices. Example quantum offerings include Amazon AWS (2023), IBM Qiskit (2023a), and Microsoft Microsoft (2023).

Most quantum offerings do not require deploying quantum circuits: to run a circuit it needs to be (i) compiled for the target quantum device and (ii) sent to the corresponding endpoint, e.g., using a specific SDK. However, the classical pre- and post-processing steps implemented in programming languages such as Python need to be deployed. This can be done using one or more cloud offerings or they can simply be executed in a local code editor when no deployment in the cloud is necessary. This choice of deployment targets is particularly important for NISQ devices, as they are limited in the number of qubits and suffer from gate and readout errors Preskill (2018). To tackle this limitation, prime NISQ use-cases focus mainly on Variational Quantum Algorithms (VQAs) Cerezo et al. (2021). VQAs comprise quantum and classical parts that are executed in a hybrid loop multiple times. Depending on how VQAs are implemented, this may result in additional integration requirements for classical and quantum parts that need to run on possibly different cloud services. Another option is to deploy quantum and classical parts together on so-called hybrid execution environments AWS (2023); Qiskit (2023a), which aim to optimize the placement of components and runtime efficiency. Hybrid execution environments, however, also impose vendor-specific implementation and packaging requirements, e.g., to use Qiskit Runtime, a quantum application must be implemented in Python and have a custom metadata file.

2.2 Patterns and Authoring Process

Patterns document abstract solutions for problems, which frequently reoccur in specific contexts Alexander et al. (1977). Networks of interconnected patterns are called pattern languages Falkenthal et al. (2018) and help solving composite problems in the corresponding domains, e.g., software engineering Coplien (1996), cloud computing Fehling et al. (2014b), or quantum computing Leymann (2019). Patterns are documented using a well-known and intuitive structure: Each pattern has a *name* and an *icon* for memorability. Further, the *problem* states the driving question for the pattern. In the *context* section, the situation where the pattern can be used is outlined. The *forces* section describes relevant conflicting factors characterizing the problem. In the *solution* section, a description is provided to show an abstract solution for the outlined situation, often accompanied by a solution sketch. Optionally, the *variants* section shows variations of the pattern. The *result* section discusses the effects of applying the pattern, i.e., the resulting context, and the *known uses* section lists distinct instances of applying this pattern in practice. Finally, the *related patterns* section documents relations with other patterns within the same language as well as relations to patterns from other pattern languages.

To systematically identify and author patterns, we used the approach by Fehling et al. (2014a), which consists of three phases: First, the *Pattern Identification* phase, in which relevant information is condensed and boundaries are identified to filter relevant information. To capture the patterns presented in this work, the documentation of different quantum cloud service providers was analyzed with respect to the capabilities they offer. Second, the *Pattern Authoring* phase, in which first the structure and format of the pattern are determined to fit the relevant domain, and then the patterns are written and iteratively revised. Finally, the *Pattern Application* phase, which is decoupled from the other two phases, involves structured search and usage of the patterns, e.g., the solutions from patterns can be applied for a specific programming language to construct a quantum circuit.

3 EXECUTION PATTERNS FOR QUANTUM APPLICATIONS

To capture different aspects of executing quantum applications, we extend the quantum computing pattern language initiated by Leymann (2019) with five new patterns. Prior to presenting the new patterns in detail, we briefly describe the quantum computing pattern language and how our extension complements it.

Figure 1 shows existing quantum computing patterns, which aim to support developers in designing and implementing quantum applications. Different aspects of the quantum application lifecycle Weder et al. (2020a) are addressed using distinct categories in the language. For example, the *Program Flow* patterns give advice on how computational tasks can be split between quantum and classical resources Weigold et al. (2021b). They explain hybrid quantum algorithms, such as Quantum Approximate Optimization Algorithm (QAOA), and describe techniques to improve them, e.g., so-called Warm-Starting. Most quantum algorithms encode classical data into the quantum circuit, which is typically done, by adding a subroutine creating the corresponding state to the beginning of the circuit. To facilitate this task, the *Data Encodings* category captures different solutions for encoding the data Weigold et al. (2020). Another aspect that must be considered is related to *Error Handling* – the eponymous category shown in Figure 1 groups various relevant patterns Beisel et al. (2022). For instance, NISQ devices can benefit from employing so-called readout error mitigation as a post-processing step to tackle measurement errors.

In this work, we introduce patterns documenting solutions for executing quantum circuits and applications that were derived by analyzing the documentation of existing quantum cloud services. Our new *Execution* patterns shown in Figure 1 aim to facilitate the decision making process for choosing quantum service offerings that fit best for given execution requirements: The STANDALONE CIRCUIT EXECUTION pattern targets the most basic execution scenarios in which quantum circuits need to be executed without any deployment or integration requirements. Thus, such scenarios can benefit from quantum offerings that simplify interaction with quantum devices. In contrast, the AD-HOC HYBRID CODE EXECUTION pattern focuses on the execution of quantum circuits with classical pre- and post-processing steps in an ad-hoc manner, i.e., without deployment customization requirements for quantum and classical code parts. More complex scenarios are covered by the PRE-DEPLOYED EXECUTION pattern in which components in quantum applications need to be deployed in a certain way, e.g., distributed to benefit from heterogeneous offerings or deployed in vicinity to increase efficiency. The PRIORITIZED EXECUTION pattern describes solutions for executing several quantum circuits efficiently by reducing the queuing times. It enables, e.g., only waiting once and then executing multiple quantum circuits with reduced waiting times or reserving a quantum device exclusively. The execution of quantum circuits as part

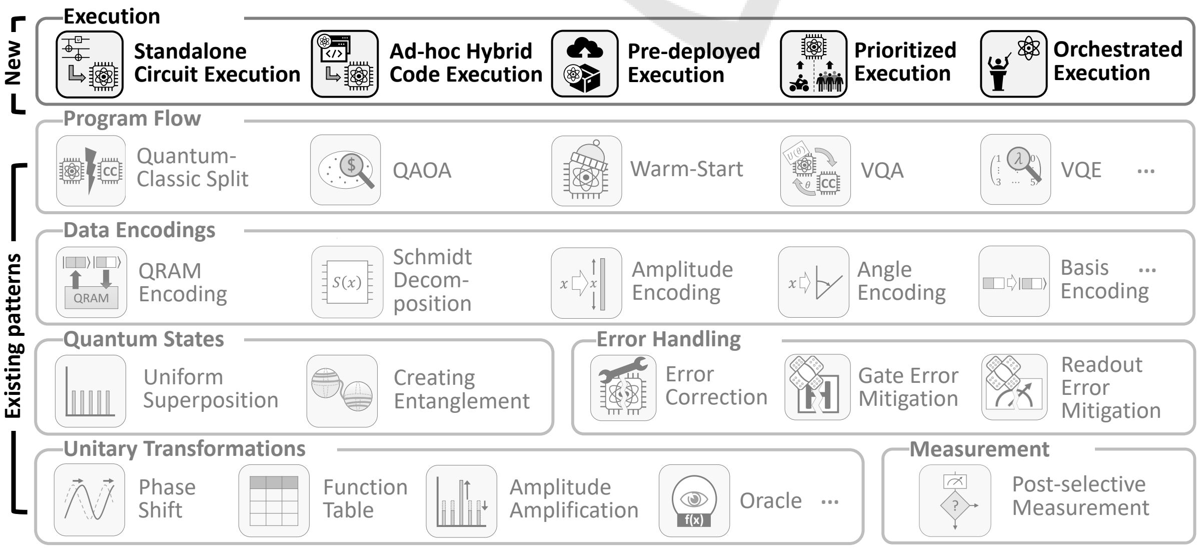

Figure 1: Overview of the quantum computing pattern language Leymann (2019); Weigold et al. (2021b); Beisel et al. (2022) with the new patterns extension proposed in this work.

of orchestrated quantum applications is described in the ORCHESTRATED EXECUTION pattern. Such applications can benefit from workflows, ensuring the data and control flow as well as providing advantages, such as scalability, robustness, or transactional processing. The STANDALONE CIRCUIT EXECUTION, AD-HOC HYBRID CODE EXECUTION, PRE-DEPLOYED EXECUTION and ORCHESTRATED EXECUTION patterns capture different execution variants for quantum applications and can be chosen based on the requirements of the given application. In contrast, the PRIORITIZED EXECUTION pattern focuses on improving the execution efficiency and can be combined with other execution patterns.

3.1 Standalone Circuit Execution

Problem: *How to execute standalone quantum circuits that impose no deployment or integration requirements?*

Context: In some scenarios, only standalone quantum circuits need to be executed, such as circuit design and testing or education. Thereby, it is beneficial to execute quantum circuits with minimal effort and required knowledge about different quantum offerings as well as deployment or integration technologies.
Forces: Quantum offerings vary in features and capabilities. Use of advanced offerings for simple use cases may incur unnecessary management overhead and even block the task due to lacking technical expertise, e.g., deployment automation and integration technologies. In contrast, certain quantum offerings reduce the amount of required management efforts, e.g., by generating execution requests automatically.
Solution: Execute quantum circuits via quantum offerings that do not require implementing custom deployment logic or integration with other services. Figure 2 shows the solution sketch in which distinct quantum circuits are executed using offerings capable of creating and executing circuits. This includes dedicated graphical circuit composers or text-based tools transmitting the quantum circuits to the offering. Typically, providers are responsible for the majority of the management efforts, e.g., deployment of circuits and authentication to the underlying quantum offering.

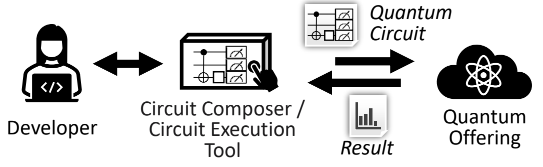

Figure 2: Solution sketch: Standalone Circuit Execution.

Result: Quantum offerings supporting this pattern provide a simple way for executing quantum circuits. However, classical pre- and post-processing steps can not be defined using these offerings. Thus, this style of execution is not suitable for running larger quantum applications and integration with external applications is either very limited or not supported at all. Additionally, running computations with this pattern cannot take advantage of prioritized executions.
Known Uses: Vendors such as IBM IBM (2023b) or Pasqal Pasqal (2023) support this pattern via graphical circuit composer services that enable modeling quantum circuits visually and executing them from the GUI, hence, abstracting away the authentication and construction of the invocation request.
Related Patterns: In scenarios when quantum circuits are executed for testing purposes and afterwards require the integration with classical pre- and post-processing steps, the AD-HOC HYBRID CODE EXECUTION can be used. Additionally, different quantum computing patterns can be used while modeling quantum circuits, e.g., AMPLITUDE AMPLIFICATION or UNIFORM SUPERPOSITION Leymann (2019).

3.2 Ad-hoc Hybrid Code Execution

Problem: *How to execute quantum circuits with classical pre- and post-processing steps with no additional deployment or integration requirements?*

Context: Often it is necessary to execute standalone quantum circuits with their classical pre- and post-processing steps without incurring extra overhead to deploy, run, and manage them in the cloud.
Forces: Similar to STANDALONE CIRCUIT EXECUTION. However, using this pattern prevents integrating the classical pre- and post-processing steps. Using local or provider-managed development environments reduces management efforts, e.g., execution of quantum and classical code via single commands.
Solution: Use provider-managed code editors or local editors with manually-added token-based authentication. Figure 3 shows the execution of quantum circuits and classical code using AD-HOC HYBRID CODE EXECUTION. The classical pre- and post-processing steps run in a local or remote code editor. The quantum circuits are executed via function calls, often provided by the programming language or quantum SDK, transmitting the quantum circuit to the quantum cloud offering.
Result: This pattern offers developers more control over the execution of quantum and classical parts, e.g., token management or request construction via SDKs, than the STANDALONE CIRCUIT EXECUTION

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

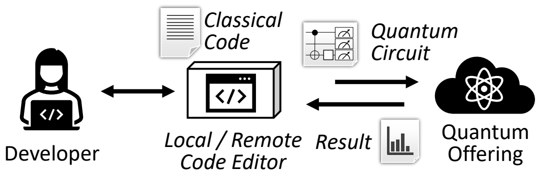

Figure 3: Solution sketch: Ad-hoc Hybrid Code Execution.

pattern. However, integration with external applications is often limited, which makes it unsuitable for designing larger applications or running several distinct computations. After a successful ad-hoc execution of quantum circuits and their classical pre- and post-processing steps, all artifacts can be packaged and deployed to a compatible offering or published via application marketplaces to simplify their reuse.

Known Uses: This pattern is supported by various vendors. For example, iPython notebooks are provided as remote code editors by Amazon with Braket AWS (2023), IBM with Qiskit IBM (2023b), and Google with Cirq Google (2020). Local code editors support different languages via dedicated packages, e.g., Rigettis pyQuil and Forest SDK Rigetti (2023a). Authentication to quantum services is then performed via manual configuration of access tokens.

Related Patterns: Classical pre-processing steps include STATE PREPARATION, which can be done using, e.g., BASIS ENCODING or ANGLE ENCODING Weigold et al. (2020). Post-processing steps comprise, e.g., READOUT ERROR MITIGATION Beisel et al. (2022). If multiple quantum circuits must be executed, PRIORITIZED EXECUTION can be used to increase efficiency.

3.3 Pre-deployed Execution

Problem: *How to execute quantum circuits with classical pre- and post-processing steps that have custom deployment requirements?*

Context: Most quantum algorithms are hybrid, e.g., VQAs contain a hybrid loop with many successive executions of parameterized quantum circuits with optimization steps in-between performed on classical hardware Cerezo et al. (2021). Further, quantum devices are accessed via the cloud and to execute a circuit, it is queued in the job-queue of the respective service, which may not support the execution of classical code parts. Thus, a quantum circuit and its classical pre- and post-processing steps may need to be deployed in a specific manner, e.g., together or using specific combinations of cloud service offerings.

Forces: Quantum service offerings vary significantly feature-wise and often rely on different authentication mechanisms, proprietary formats, and SDKs. Additional technical expertise may be required to successfully execute a provided quantum circuit with its pre- and post-processing steps that can be hosted separately, e.g., due to data processing requirements. In certain cases, it is more beneficial to execute quantum and classical parts of the application in proximity of each other, e.g., to reduce the networking overhead.

Solution: Pre-deploy the quantum circuit with its pre- and post-processing steps either on (i) a single quantum offering that supports execution of quantum and classical parts, or (ii) a specific combination of quantum and cloud offerings that fulfills the given deployment requirements. Figure 4 shows the solution sketch of this pattern: In *Step 1*, the quantum and classical parts are deployed according to deployment preferences. The subsequent execution and fetching of the results shown in *Steps 2&3* can be done independently by developers or client applications.

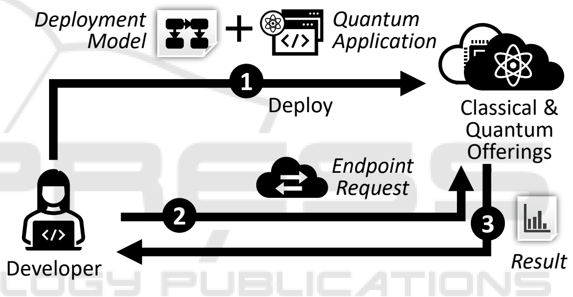

Figure 4: Solution sketch for the Pre-deployed Execution pattern: a quantum application is executed on a combination of different cloud offerings.

Variants: One deployment target option in *Step 1* is a *Hybrid Execution Environment* offering that can speed up the interaction between quantum and classical computations. Hence, they reduce the network overhead and queuing times for many successive quantum circuit executions. Another variant is a *Distributed Deployment* in which parts of the quantum application are deployed on a combination of different cloud offerings. Implementation of quantum and classical parts, as well as their deployment models, depend on chosen technologies, e.g., implementation of QAOA for Qiskit Runtime would impose more coding and deployment modeling constraints compared to more general deployment scenarios such as packaging the quantum application as one or more containers that can be deployed to a container orchestration engine such as Kubernetes and later executed via client requests.

Result: The deployment of quantum applications is decoupled from its execution, hence, enabling the

invocation by other users or integration with other applications, e.g., a pre-deployed VQA can be subsumed as a part of another application. In the case of hybrid runtimes, the pre-deployed application benefits from provider-managed execution transparency but is locked into the requirements and limitations of the underlying offering. In the Distributed Deployment variant, developers can benefit from combining different services for specific parts of the application.
Known Uses: Qiskit Runtime Qiskit (2023a) is a service from IBM, which offers a hybrid execution environment that enables deploying a quantum application packaged as a Python file with a JSON metadata file, and subsequently executing it via a publicly available HTTP endpoint. A similar service is Amazon Braket Hybrid Jobs AWS (2023). Microsoft also introduced the hybrid execution environment and shows an example how to use it Frachon (2023). For distributed deployment scenarios, various cloud offerings can be employed for classical tasks. Example offerings from AWS include *AWS Lambda* for executing classical Python code, *AWS S3* as an object storage offering, and *Amazon Cloudwatch* for monitoring.
Related Patterns: Pre-deployed execution of hybrid applications improves the reusability of different algorithms such as the VQA and the more concrete patterns like VQE and QAOA Weigold et al. (2021b). Additionally, pre- and post-processing steps, such as STATE PREPARATION Leymann (2019) or READOUT ERROR MITIGATION Beisel et al. (2022), can be applied to run on preferred execution targets.

3.4 Prioritized Execution

Problem: *How to execute multiple quantum circuits in succession while keeping the queuing time low?*

Context: Quantum applications often require executing multiple quantum circuits. This is especially the case when utilizing VQAs on contemporary NISQ devices. Thus, these quantum circuits should be executed efficiently by minimizing the queuing times.
Forces: Quantum devices are usually accessed via queues, ensuring their fair utilization. Thus, the queuing times sum up when executing multiple quantum circuits independently of each other. One approach is to use batch processing, i.e., combining various quantum circuits into one job, which is then executed at once Vietz et al. (2021). However, this approach is not possible when quantum circuits depend on the results of previous executions, e.g., for VQAs.
Solution: Use quantum offerings that enable prioritized access to quantum devices to reduce or completely avoid queuing times, as shown in the solutions sketch in Figure 5. For this, prioritized access to quantum devices is obtained in *Step 1*. In *Step 2*, the quantum device can then be reused via this prioritized access, which restricts the number of users to reduce queuing times for multiple circuit executions.

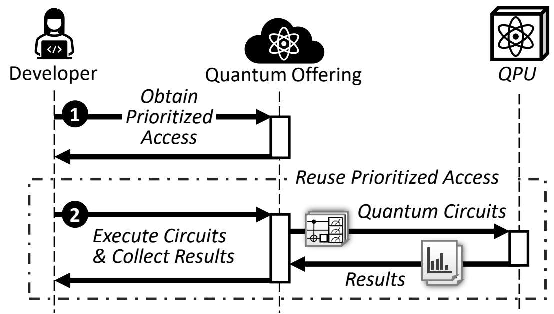

Figure 5: Solution sketch for Prioritized Execution showing multiple circuits executed via prioritized access.

Variants: One variant is to enable prioritization via *Session Reuse*. Hence, the session established for the first quantum circuit execution is reused for multiple subsequent circuit executions. *Dedicated Access* is another variant in which a quantum device is reserved for exclusive use without initial queuing by booking a time-slice. However, this typically incurs additional costs.
Result: By applying this pattern, the time spent waiting inside highly occupied job queues is minimized. The choice between the two options is a trade-off between runtime and cost. In the reserved time-slice scenario, runtime is the highest priority since the reservation of a quantum device incurs higher costs than other forms of access. When using sessions for the execution, there can still be competing executions in the job queue from other users.
Known Uses: The Session Reuse variant is currently supported by IBM Qiskit (2023a) and AWS AWS (2023). Further, the Dedicated Access variant is offered by vendors such as AWS AWS (2023), Azure Microsoft (2023), or IBM IBM (2023b). A concrete example of a prioritized execution using Qiskit Qiskit (2023b) highlights which options for the session are available and shows the details like the session time limit.
Related Patterns: When using the VQA, VQE, or QAOA patterns Weigold et al. (2021b), prioritization is highly advised because of the amount of sequential quantum circuit executions. In these cases, it can be combined with the PRE-DEPLOYED EXECUTION pattern to address custom deployment requirements.

3.5 Orchestrated Execution

Problem: *How to ensure the control and data flow for quantum applications comprising one or more quantum circuits with corresponding classical pre- and post-processing steps?*

Context: Most quantum algorithms are hybrid, i.e., parts are executed on quantum devices, and others run on classical hardware Leymann and Barzen (2020). Furthermore, quantum applications can involve multiple quantum algorithms and additional classical parts, e.g., interacting with the user or loading data from a database. These parts must be orchestrated, i.e., the control and data flow between them must be ensured.
Forces: Quantum devices and the corresponding quantum cloud offerings vary strongly in characteristics, such as the number of available qubits, incurred costs for the execution, or queuing times Tannu and Qureshi (2019); Vietz et al. (2021). The orchestration of parts running in heterogeneous environments can get unmanageable without external orchestration tools, e.g., as there could be long invocation chains, complex data transfers, data format transformations, and interactions with various heterogeneous APIs.
Solution: Utilize a workflow language to model the quantum and classical parts as tasks within a workflow model Weder et al. (2020b) as shown in Figure 6. The workflow model can then be deployed to a workflow engine, which orchestrates the quantum and classical parts by invoking them in the specified order and ensuring the required data flow Ellis (1999). Thereby, the invocation of heterogeneous offerings, as well as features such as data format transformation is provided by the workflow engine Leymann and Roller (2000). While there exist workflow offerings specifically targeting the quantum computing domain, e.g., providing some pre-implemented quantum algorithms, standardized workflow languages and corresponding workflow engines can also be employed to benefit from their maturity and rich feature sets.
Result: The classical code as well as the quantum circuits required to realize a quantum application are separated from the workflow model defining how they are integrated. This increases modularity and enables the reuse of existing code, decreasing development time and cost. Furthermore, by using workflows, quantum applications can benefit from the reliability, scalability, and robustness of workflow engines Leymann and Barzen (2021a). Finally, also the usage of various heterogeneous quantum and classical cloud offerings with different functionalities is supported. However, the need to model orchestrations to enact them on specialized middleware requires additional expertise and may result in overhead for simple use cases such as circuit design and testing.
Known Uses: Examples of standardized workflow languages are *Business Process Model and Notation (BPMN)* Object Management Group (2010) or *Business Process Execution Language (BPEL)* OASIS (2007). For the execution of these workflow models, different workflow engines are available, e.g., the Camunda BPMN engine Camunda Services (2023).

Furthermore, there exists a quantum-specific modeling extension to ease the modeling of workflows in the quantum computing domain Weder et al. (2020b). Quantum-specific orchestration tools comprise *Orquestra* by Zapata Zapata (2022), which uses a custom YAML-based language, and *Covalent* by Agnostiq Agnostiq (2023), a quantum orchestration platform that requires specifying control flow via a Python-based domain-specific language.
Related Patterns: If multiple quantum circuits have to be executed by the workflow, the PRIORITIZED EXECUTION pattern can be used to reduce waiting times. Further, this pattern can be applied to integrate several distinct quantum applications made available using the PRE-DEPLOYED EXECUTION pattern.

4 PATTERNS SUPPORT IN QUANTUM OFFERINGS

This section provides an overview of existing quantum cloud service offerings for which documentation has been analyzed to extract the patterns introduced in Section 3. To compile a list of cloud services relevant for this work, we conducted a thorough search of relevant literature Fingerhuth et al. (2018); Gill et al. (2020), and related websites Dilmegani (2023); PAT Research (2023); Quantum Computing Report

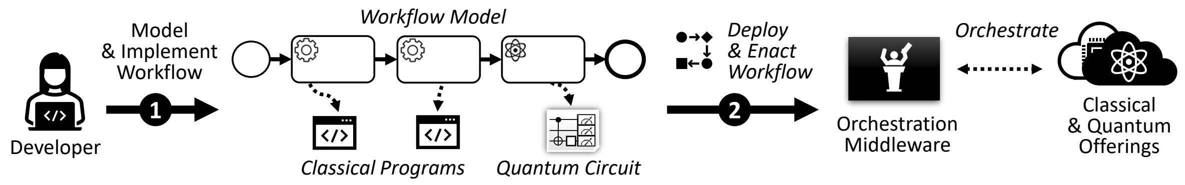

Figure 6: Solution sketch for an orchestrated execution. Implemented quantum and classical parts are tasks that are executed in the specified order and environment with the help of an orchestration middleware.

Table 1: Patterns Support in Quantum Cloud Offerings: (✓) supported, (✗) not documented.

	STANDALONE CIRC. EXEC.	AD-HOC HYBRID CODE EXEC.	PRE-DEPLOYED EXECUTION		PRIORITIZED EXECUTION	
			Hybrid Runtime	Distributed Deployment	Session Reuse	Dedicated Access
AWS (2023)	✓	✓	✓	✓	✓	✗
Google (2023)	✓	✓	✗	✓	✗	✗
IBM (2023a)	✓	✓	✓	(✓)[1]	✓	✓
IonQ, Inc. (2023)	✓	✓	✗	✗	✗	✓
Microsoft (2023)	✓	✓	✓	✓	✗	✗
Pasqal (2023)	✓	✓	✗	✗	✗	✗
QC Ware (2023)	✓	✓	✗	✗	✗	✗
Quantastica (2023)	✓	✓	✗	✗	✗	✗
QuTech (2023)	✓	✓	✗	✗	✗	✗
Rigetti (2023b)	(✓)[2]	✗	(✓)[2]	✗	✗	(✓)[2]
Xanadu (2023)	✓	✓	✗	✗	✗	✓

[1] Requires multi-cloud setup (classical and quantum services of the provider are separated)
[2] Partially supported, similar experience is achieved via obligatory VPN connections

(2023). The resulting list encompasses a diverse range of quantum cloud services, as well as various projects and libraries that can be utilized for development purposes. Additionally, we analyzed which cloud services are used or supported by the initial list of the identified projects and libraries to explore further cloud services that were not initially identified in our search. To identify cloud services' support for the patterns and their variants, we analyzed the documentation of the respective cloud services and explored the services directly if publicly accessible.

Table 1 presents a summary of our findings. Unsurprisingly, the STANDALONE CIRCUIT EXECUTION pattern (see Section 3.1) is supported by all vendors. Typically, this is accomplished via public API endpoints that receive quantum circuits. Some cloud providers have integrated a graphical circuit composer into their offering for this purpose. Besides graphical circuit composers, which are also available in countless variations as standalone SaaS services, circuits can be constructed manually or using other tools. Please, note that different cloud services expect the circuits in different formats. The AD-HOC HYBRID CODE EXECUTION pattern (see Section 3.2) is also supported by almost all cloud providers. Cloud providers enable this type of execution by special SDKs that can be used for programming and execution. While some cloud providers (e.g., AWS, IBM, Microsoft) develop their own custom SDKs, other hardware vendors (e.g., IonQ) provide plugins for third-party SDKs through which their service offerings can be used. For the PRE-DEPLOYED EXECUTION pattern (see Section 3.3), we analyzed quantum service offerings with respect to their support for both variants. The first variant is the *Hybrid Runtime*, which is a specialized service that can execute both quantum and classical components that are closely interconnected. The second variant is a deployment on classical services, where the deployed applications communicate with quantum-specific services. Table 1 shows that the *Hybrid Runtime* is supported by four providers. For single-cloud deployments, the second variant is basically supported by all cloud services that offer classical services, i.e., Microsoft Azure, Amazon Web Services, and IBM cloud. Unlike the mentioned cloud providers, the rest have only quantum-specific services in their portfolio, which implies modeling multi-cloud deployments and may result in extra integration requirements.

The PRIORITIZED EXECUTION pattern can help to avoid long waiting times when executing multiple quantum circuits by allowing users to prioritize their requests. As discussed in Section 3.4, we have identified two variants for this pattern: *Session Reuse*, and *Dedicated Access*. Besides Rigetti QCS Rigetti (2023b) which requires a VPN connection for deployment, *Session Reuse* is offered by the same providers that also offer a *Hybrid Runtime* as it inherently enables this type of prioritized execution. A minor part of analyzed services offer a dedicated access to quantum devices: To get this access, some providers need to be contacted directly first, as there is often no predefined booking action through interfaces.

The ORCHESTRATED EXECUTION pattern (see Section 3.5) is enabled by quantum-specific workflow technologies which provide means for automating the execution of quantum and classical tasks. Solutions can be divided into two categories: standards-

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

based and non-standards-based solutions. For example, BPMN or BPEL can be used to model and execute quantum workflows Weder et al. (2020b). Non-standards-based solutions include Orquestra Zapata (2022) and Covalent Agnostiq (2023).

5 RELATED WORK

This section elaborates on the related research publications and aligns our contributions with respect to them. The execution patterns for quantum applications presented in this work extend the existing quantum computing pattern language Leymann (2019); Weigold et al. (2021a,b); Beisel et al. (2022). Although there are other papers publishing patterns for quantum computing Gilliam et al. (2019); Perdrix (2007) and validating patterns in quantum circuits Huang and Martonosi (2019), they do not follow the pattern format of Alexander et al. (1977). To the best of our knowledge, there are no other works introducing patterns in the quantum computing domain.

The patterns presented in this work are not limited to concepts from the quantum computing domain, rather they are interdisciplinary, and also utilize well-established concepts from other domains, such as the cloud computing and application integration. Fehling et al. (2014b) present a pattern language for cloud computing, describing solutions to commonly reoccurring problems of cloud applications. For example, the quantum cloud offerings used in this work are related to the PLATFORM AS A SERVICE and SOFTWARE AS A SERVICE patterns. Hohpe and Woolf (2004) introduce the Enterprise Integration Patterns (EIP), which support developers in building distributed and integrated applications. The PROCESS MANAGER pattern is part of the EIPs and is closely related to the ORCHESTRATED EXECUTION pattern introduced in this work, as both patterns describe how a multitude of programs can be orchestrated by a central unit. In contrast to the PROCESS MANAGER pattern, the ORCHESTRATED EXECUTION pattern focuses on requirements imposed by the orchestration and integration of quantum applications.

To facilitate the visualization of links between patterns within a pattern language and advocate the interconnection of patterns across different domains, Leymann and Barzen (2021b) introduce the Pattern Atlas. The Pattern Atlas is a publicly available, cartography-inspired tool PlanQK (2023), which enables the creation of new connections between different pattern languages as well as *Pattern Views*, i.e., subsets of patterns and connections from different languages.

Different research works introduce concepts for executing and deploying quantum applications. Nguyen et al. (2022) present a framework, which enables the execution of quantum applications in a serverless environment. Weder et al. (2022) provide a concept to rewrite quantum workflows for the pre-deployed execution in hybrid runtimes to increase their efficiency. Leymann and Barzen (2021a) propose an approach for the orchestration and deployment of hybrid quantum applications using workflows as well as deployment automation technologies.

6 SUMMARY AND OUTLOOK

Quantum devices and services evolve rapidly and with them, real use-cases are in sight. In this work, we captured different possibilities to use quantum devices and documented them as five new patterns which extend the quantum computing patterns language. The proposed patterns capture alternatives on how to execute quantum applications of different granularity from atomic components like a quantum circuit to fully orchestrated quantum workflows. We also analyzed and discussed which service providers support the execution methods documented in our patterns. All listed quantum service providers support basic circuit executions described in the STANDALONE CIRCUIT EXECUTION and the AD-HOC HYBRID CODE EXECUTION. Support for patterns such as the PRE-DEPLOYED EXECUTION and PRIORITIZED EXECUTION varies among the analyzed quantum offerings. Additionally, we discuss several tools for the orchestration of quantum applications enabling the ORCHESTRATED EXECUTION pattern.

Since patterns always evolve, we plan to continue revising the introduced patterns and document new execution alternatives that could appear in the future. We also plan to enhance the patterns presented in this work with more examples referencing concrete use cases and to evaluate the advantages and disadvantages of applying the patterns in different scenarios. Furthermore, we intend to incorporate the proposed patterns in the pattern landscape within the Pattern Atlas PlanQK (2023) and connect them to existing patterns that benefit from a certain style of execution. Additionally, we plan to further extend the quantum computing patterns language by analyzing which best practices appear in quantum software engineering. Another direction is to automate the decision making process for selecting which execution patterns fit best for a given scenario.

ACKNOWLEDGEMENTS

This work was partially funded by the BMWK projects *SeQuenC* (01MQ22009B), *EniQmA* (01MQ22007B), and *PlanQK* (01MK20005N).

REFERENCES

Agnostiq (2023). Covalent. https://agnostiq.ai/covalent/.

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

Arute, F., Arya, K., Babbush, R., Bacon, D., et al. (2019). Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510.

AWS (2023). Amazon Braket Usage Overview. https://docs.aws.amazon.com/braket/latest/developerguide/braket-using.html.

Barzen, J. (2021). From Digital Humanities to Quantum Humanities: Potentials and Applications. In *Quantum Computing in the Arts and Humanities*. Springer. arXiv:2103.11825.

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 2022)*, pages 22–30. Xpert Publishing Services (XPS).

Camunda Services (2023). Camunda workflow engine. https://docs.camunda.io.

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.

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., McClean, J. R., Mitarai, K., Yuan, X., Cincio, L., and Coles, P. J. (2021). Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644.

Coplien, J. O. (1996). *Software Patterns*. SIGS Books & Multimedia.

Dilmegani, C. (2023). Quantum Software in 2023: What It Is & How It Works. https://research.aimultiple.com/quantum-software/.

Ellis, C. A. (1999). Workflow Technology. *Computer Supported Cooperative Work, Trends in Software Series*, 7:29–54.

Falkenthal, M., Breitenbücher, U., and Leymann, F. (2018). The Nature of Pattern Languages. In *Pursuit of Pattern Languages for Societal Change*, pages 130–150. tredition.

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 2014)*. 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.

Fingerhuth, M., Babej, T., and Wittek, P. (2018). Open source software in quantum computing. *PloS one*, 13(12):1–28.

Frachon, F. (2023). Azure Quantum unlocks the next generation of Hybrid Quantum Computing. https://devblogs.microsoft.com/qsharp/azure-quantum-unlocks-the-next-generation-of-hybrid-quantum-computing/.

Gabor, T., Sünkel, L., Ritz, F., Phan, T., Belzner, L., Roch, C., Feld, S., and Linnhoff-Popien, C. (2020). The Holy Grail of Quantum Artificial Intelligence: Major Challenges in Accelerating the Machine Learning Pipeline. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 456–461.

Gill, S. S., Kumar, A., Singh, H., Singh, M., Kaur, K., Usman, M., and Buyya, R. (2020). Quantum Computing: A Taxonomy, Systematic Review and Future Directions. arXiv:2010.15559.

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.

Google (2020). Google cirq overview. https://colab.research.google.com/github/quantumlib/Cirq/blob/master/docs/tutorials/google/start.ipynb.

Google (2023). Google Quantum AI. https://quantumai.google/.

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*, pages 541–553.

IBM (2023a). IBM Quantum. https://quantum-computing.ibm.com.

IBM (2023b). IBM Quantum Lab. https://quantum-computing.ibm.com/lab/docs/iql/.

IonQ, Inc. (2023). IonQ. https://ionq.com/.

LaRose, R. (2019). Overview and Comparison of Gate Level Quantum Software Platforms. *Quantum*, 3.

Leymann, F. (2019). Towards a Pattern Language for Quantum Algorithms. In *First International Workshop, QTOP 2019, Proceedings*. 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. (2021a). Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective. arXiv:2103.04320.

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

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 2020)*, pages 9–24. SciTePress.

Leymann, F. and Roller, D. (2000). *Production Workflow: Concepts and Techniques*. Prentice Hall PTR.

Microsoft (2023). Microsoft Azure Quantum. https://azure.microsoft.com/en-us/services/quantum/.

Nguyen, H. T., Usman, M., and Buyya, R. (2022). QFaaS: A Serverless Function-as-a-Service Framework for Quantum Computing.

OASIS (2007). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS).

Object Management Group, I. O. (2010). BPMN 2.0. https://www.omg.org/spec/BPMN/2.0/PDF.

Pasqal (2023). Pasqal. https://pasqal.io.

PAT Research (2023). What is Quantum Computing? Top 18 Quantum Computing Companies. https://www.predictiveanalyticstoday.com/what-is-quantum-computing/.

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

PlanQK (2023). PlanQK - Integration of the patternrepository pattern atlas. https://patterns.platform.planqk.de/pattern-languages.

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

QC Ware (2023). QC Ware Forge. https://forge.qcware.com.

Qiskit (2023a). Qiskit: An Open-source Framework for Quantum Computing. https://qiskit.org/documentation/.

Qiskit (2023b). Qiskit: Run a primitive in a session. https://qiskit.org/documentation/partners/qiskit_ibm_runtime/how_to/run_session.html.

Quantastica (2023). Quantum Programming Studio. https://quantum-circuit.com.

Quantum Computing Report (2023). Tools. https://quantumcomputingreport.com/tools/.

QuTech (2023). Quantum Inspire. https://www.quantum-inspire.com/.

Rigetti (2023a). PyQuil and Forest SDK. https://pyquil-docs.rigetti.com/en/v3.3.3/.

Rigetti (2023b). Rigetti Quantum Cloud Services. https://docs.rigetti.com/qcs/.

Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A., and Duncan, R. (2020). t|ket⟩: A retargetable compiler for NISQ devices. *Quantum Science and Technology*.

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*, pages 987–999.

Vietz, D., Barzen, J., Leymann, F., Weder, B., and Yussupov, V. (2021). An Exploratory Study on the Challenges of Engineering Quantum Applications in the Cloud. In *Proceedings of the 2nd Quantum Software Engineering and Technology Workshop (Q-SET 2021) co-located with IEEE International Conference on Quantum Computing and Engineering (QCE21)*, pages 1–12. CEUR Workshop Proceedings.

Weder, B., Barzen, J., Beisel, M., and Leymann, F. (2022). Analysis and Rewrite of Quantum Workflows: Improving the Execution of Hybrid Quantum Algorithms. In *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER 2022)*, pages 38–50. SciTePress.

Weder, B., Barzen, J., Leymann, F., Salm, M., and Vietz, D. (2020a). The Quantum Software Lifecycle. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*, pages 2–9. ACM.

Weder, B., Breitenbücher, U., Leymann, F., and Wild, K. (2020b). Integrating Quantum Computing into Workflow Modeling and Execution. In *Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020)*, pages 279–291. IEEE Computer Society.

Weigold, M., Barzen, J., Leymann, F., and Salm, M. (2020). Data Encoding Patterns For Quantum Algorithms. In *Proceedings of the 27th Conference on Pattern Languages of Programs (PLoP '20)*, pages 1–11. HILLSIDE.

Weigold, M., Barzen, J., Leymann, F., and Salm, M. (2021a). Expanding Data Encoding Patterns For Quantum Algorithms. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pages 95–101. IEEE.

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 2021)*, pages 34–51. Springer International Publishing.

Xanadu (2023). Xanadu Quantum Cloud. https://www.xanadu.ai/cloud.

Zapata (2022). Orquestra. https://www.zapatacomputing.com/orquestra.

All links were last followed June 10, 2023.