SN Computer Science (2023) 4:233
https://doi.org/10.1007/s42979-022-01625-9

ORIGINAL RESEARCH

Provenance-Preserving Analysis and Rewrite of Quantum Workflows for Hybrid Quantum Algorithms

Benjamin Weder[1] · **Johanna Barzen**[1] · **Martin Beisel**[1] · **Frank Leymann**[1]

Received: 29 August 2022 / Accepted: 20 December 2022 / Published online: 23 February 2023

Abstract

Quantum applications are hybrid, i.e., they comprise quantum and classical programs, which must be orchestrated. Workflows are a proven solution for orchestrating heterogeneous programs while providing benefits, such as robustness or scalability. However, the orchestration using workflows can be inefficient for some quantum algorithms, requiring the execution of quantum and classical programs in a loop. Hybrid runtimes are offered to efficiently execute these algorithms. For this, the quantum and classical programs are combined in a single hybrid program, for which the execution is optimized. However, this leads to a conceptual gap between the modeling benefits of workflow technologies, e.g., modularization, reuse, and understandability, and the efficiency improvements when using hybrid runtimes. To close this gap, we introduce a method to model all tasks explicitly in the workflow model and analyze the workflow to detect parts of the workflow that can benefit from hybrid runtimes. Furthermore, corresponding hybrid programs are automatically generated based on the quantum and classical programs, and the workflow is rewritten to invoke them. To ease the live monitoring and later analysis of workflow executions, we integrate process views into our method and collect related provenance data. Thus, the user can visualize and monitor the workflow in the original and rewritten form within the workflow engine. The practical feasibility of our approach is validated by a prototypical implementation, a case study, and a runtime evaluation.

Keywords Quantum computing · Hybrid algorithms · Quantum workflows · Workflow rewrite · Hybrid runtimes · Process views

Introduction

With recent advances in improving quantum computers, e.g., increasing the number of provided qubits or reducing their error rates, quantum applications from various application areas, such as physics or chemistry, can be implemented and executed on real quantum computers [3, 89]. Furthermore, the quantum computers are publicly accessible via the cloud, e.g., by providers like IBM, Rigetti, or IonQ, enabling their usage by a large user group [42, 48, 83]. In addition to the speed-up that is proven for different quantum algorithms compared to their best known classical counterpart, there are further benefits for some use cases, such as a higher accuracy for some machine learning algorithms or a lower energy consumption when solving the same problem [5, 27].

In the following, we restrict ourselves to the widely used gate-based quantum computing model [46, 57, 60]. Thereby, quantum programs are represented by *quantum circuits*, comprising a set of *gates* operating on qubits and *measurements* to retrieve classical information from them [42, 48]. However, quantum applications do not only consist of quantum programs. Instead, they are hybrid, i.e., they also comprise classical programs, e.g., performing pre-and post-processing tasks [55, 89]. One of these classical pre-processing tasks is generating so-called *state preparation circuits* based on the input data, which are then added to the beginning of the original circuit to prepare the required state in the register of the quantum computer [43, 46]. On the other hand, a classical post-processing task is *error mitigation*, which is used to reduce the impact of errors occurring due to the noise of current quantum computers [7, 54]. Finally, some

This article is part of the topical collection "Advances on Cloud Computing and Services Science" guest edited by Donald F. Ferguson, Claus Pahl and Maarten van Steen.

✉ Benjamin Weder
benjamin.weder@iaas.uni-stuttgart.de

[1] Institute of Architecture of Application Systems (IAAS), University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany

classical tasks are also independent of a certain quantum algorithm, such as loading data from a database or visualizing results for the user [47, 89].

However, all the required quantum and classical programs are often based on varying programming languages, invocation mechanisms, or data formats [47, 83]. Thus, orchestrating their control and data flow is complex [91]. *Workflow technologies* are a proven solution that has been used for the integration and orchestration of heterogeneous applications in different areas, e.g., business process management or e-science [51, 52]. By providing various advantages, such as scalability, reliability, robustness, automated error handling, or transactional processing, workflows are a promising means to orchestrate the quantum and classical programs of a hybrid quantum application [22, 45, 47]. Furthermore, there exist already modeling extensions to ease the modeling of such quantum workflows [91].

Although workflows provide a variety of benefits when modeling hybrid quantum applications, the orchestration can get inefficient for some use cases. This is the case if a high number of iterations between classical and quantum processing is needed, as the data transfer times, as well as the queuing times, sum up then [47]. One special type of quantum algorithm requiring these iterations are *variational algorithms* performing the interleaved execution of quantum and classical programs until a certain condition is met [24, 56]. These variational algorithms can be efficiently executed by so-called *hybrid runtimes*, enabling to upload all quantum and classical programs together as *hybrid program* and optimizing the communication between the quantum computer and the classical infrastructure [83].

In this paper, (1) we present a method to benefit from the advantages of modeling hybrid quantum applications using workflows while increasing the efficiency by utilizing hybrid runtimes for appropriate workflow parts. Thus, the user can model the workflow comprising all required tasks, retaining the modularity, increasing the reusability, and easing the exchange of implementations for certain tasks. However, (2) the workflow model is automatically analyzed before execution to find suitable workflow parts that can benefit from hybrid runtimes. Then, corresponding hybrid programs are generated, and the workflow is rewritten to invoke them instead of orchestrating the whole workflow part. Monitoring and analyzing the rewritten workflow is challenging for the user, as it differs from the original workflow. To overcome this issue, (3) we automatically collect provenance data about the running hybrid programs, such as the current state or intermediate results, which are used to provide process views to the user, allowing to visualize the original and the rewritten workflow with its current state in the workflow engine. To prove the practical feasibility of our concepts, (4) we present a prototypical implementation based on the *MODULO framework* [86], as well as *Qiskit Runtime* [33] and *Amazon Braket Hybrid Jobs* [4], two hybrid runtimes provided by IBM and AWS. Finally, (5) we validate our concepts based on exemplary quantum workflows and evaluate the achieved runtime advantages for rewritten workflows.

This paper is a substantially extended and revised version of [85]. The main changes are: Improving and extending the introduction, complementing the fundamentals with information about provenance and process views, and adding a second research question to the problem statement. Furthermore, the analysis and rewrite method from section "Quantum Workflow Analysis and Rewrite" was extended with two new steps related to the provenance preservation when applying our method. Section "Monitoring & Analysis of Rewritten Quantum Workflows" was newly added to discuss more details about process views for quantum workflows and the retrieval of provenance data from hybrid runtimes. Additionally, the prototype, as well as the case study, were extended to cover the newly added concepts. Finally, the evaluation was enlarged with new experiments, and related work by further papers concerning provenance and process views.

The remainder of this paper is organized as follows: section "Fundamentals & Problem Statement" introduces the required background and the problem statement that underlies our work. In section "Quantum Workflow Analysis and Rewrite", the method to model quantum workflows independent of a certain runtime, as well as their automated analysis and rewrite to improve the efficiency of the execution, is presented. Next, section "Monitoring & Analysis of Rewritten Quantum Workflows" shows how to retrieve provenance data from hybrid runtimes, and how process views can be used to properly monitor and analyze rewritten workflows. Section "Prototypical Validation" introduces the architecture and the prototypical implementation of the system realizing the proposed concepts. In section "Case Study", the concepts are validated with a case study, and section "Evaluation" evaluates the performance when applying our method and compares the runtimes of original and rewritten workflows. Finally, section "Discussion" discusses possible extensions to our concepts, section "Related Work" presents the related work, and section "Conclusion & Future Work" concludes the paper.

Fundamentals and Problem Statement

In this section, the required background about hybrid quantum algorithms and hybrid runtimes, workflows to orchestrate hybrid quantum applications, and provenance to monitor and analyze the workflows are introduced. Finally, we discuss the problem statement of this work and present the corresponding research questions.

Hybrid Quantum Algorithms and Hybrid Runtimes

Although quantum computing is rapidly evolving, today's quantum computers are still restricted [46, 64]. On the one hand, they provide only a limited number of qubits, restricting the size of the input data that can be processed, e.g., the number to factorize for *Shor's algorithm* [17, 71]. On the other hand, noise from different sources, such as unintended interactions between the qubits, leads to high error rates. This limits the maximum depth, i.e., the number of sequential operations on qubits, of successfully executable quantum circuits [75]. Thus, they are often referred to as *Noisy Intermediate-Scale Quantum (NISQ)* computers [65]. However, with these restrictions, some quantum algorithms, that provide a speed-up compared to their classical counterparts can not be executed on practically useful problems [46].

Therefore, so-called *hybrid quantum algorithms*, or *hybrid algorithms* for short, are used to already benefit from quantum computing even with today's restricted computers [47, 56, 89]. For this, they combine calculations on quantum and classical computers to solve the overall problem [92]. By utilizing this approach, the size of the quantum programs to execute can be reduced. This means they only require a small number of qubits and operations, and hence, the quantum programs can be successfully executed on today's quantum computers [14, 46]. In addition, different hybrid algorithms execute the quantum and classical programs interleaved multiple times, e.g., variational algorithms perform classical optimization and quantum program execution in a loop [56]. This approach is realized by widely used algorithms, such as the *Variational Quantum Eigensolver (VQE)* [36] to determine eigenvalues of a matrix or the *Quantum Approximate Optimization Algorithm (QAOA)* [24] to solve optimization problems.

Quantum computers are typically accessed through a queue, i.e., a job is put in the queue, then it waits for execution, and finally, the results can be retrieved [37, 47]. This is unsuitable when hybrid algorithms require multiple interleaved quantum and classical computations, as the waiting times sum up [53]. A possible solution is the reservation of a time slice for the exclusive execution on a quantum computer [42]. However, it is difficult to estimate the required execution times, which can lead to unused reservations and corresponding costs [83]. Additionally, the latency incurred by transmitting the data between the quantum computer and the classical execution environment, e.g., a local computer or some cloud offering, still increases the overall execution time [31]. Thus, different providers develop new offerings optimizing the execution of such algorithms, to which we refer to as hybrid runtimes. Examples of hybrid runtimes are *Qiskit Runtime* [33] by IBM or *Amazon Braket Hybrid Jobs* [4] by AWS. Thereby, the quantum and classical programs implementing a hybrid algorithm can be uploaded together to the hybrid runtime [83]. The hybrid runtime optimizes the data transfer time by closely deploying the classical programs to the used quantum computer. Further, after the initial queue access, the hybrid program starts and has preferred or exclusive access to the quantum computer, reducing intermediate queuing times [4, 31].

Quantum Workflows and QuantME

Due to the heterogeneity of the quantum and classical programs implementing a hybrid algorithm, the orchestration of the control and data flow between them can be challenging [86, 91]. This problem is amplified for complex hybrid quantum applications, as they might comprise multiple hybrid algorithms and further classical programs independent of a quantum algorithm [47]. For this orchestration, workflow technologies should be used to benefit from their advantages, e.g., scalability, reliability, transactional processing, or the possibility to interrupt a long-running workflow execution [23, 51]. Thereby, a *workflow model* is defined, comprising a collection of activities, as well as specifying their partial order and the data flow between them. These workflow models can then automatically be executed by a workflow engine, navigating through the workflow model, transferring and, if needed, transforming the data, and triggering the execution of the different programs [49].

Existing workflow languages, such as the *Business Process Model and Notation (BPMN)* [62] or the *Business Process Execution Language (BPEL)* [61], do not provide a means to explicitly model the execution of quantum programs [86, 91]. Technically, workflows enable modeling these tasks, as arbitrary kinds of tasks can be defined. However, this leads to a complex modeling process that requires a lot of quantum-specific expertise [83, 89]. To ease the modeling of quantum workflows and to increase the reuse of corresponding implementations, we introduced the *Quantum Modeling Extension (QuantME)* [91] for imperative workflow languages. QuantME provides explicit modeling constructs for different frequently occurring pre-processing, quantum program execution, and post-processing tasks. For example, it includes the *data preparation task* to generate a state preparation circuit based on given input data or the *quantum circuit execution task* to execute quantum circuits. Furthermore, each task has a set of configuration properties to customize them depending on the requirements, e.g., by specifying the quantum computer to use for the execution of a quantum circuit.

Figure 1 shows an exemplary quantum workflow comprising the general structure of variational algorithms, such as VQE or QAOA [14, 92]. When a message is received, the workflow is instantiated, and the algorithm is initialized

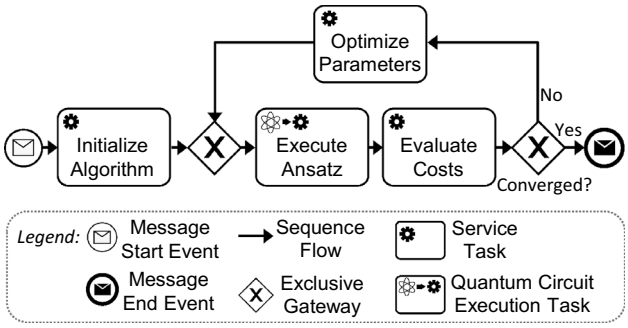

Fig. 1 Quantum workflow executing a variational algorithm [85]

by loading the *ansatz*, a parameterized quantum circuit for which the parameters are optimized in multiple iterations [25, 72]. Then, a hybrid loop is entered with the initial parameters, first executing the ansatz. Based on the execution results, a cost function encoding the solutions of the problem to solve is evaluated. Next, the gateway checks if the costs converge or another termination condition is met, and in this case the workflow terminates and sends the result to the user. Otherwise, the next iteration starts by optimizing the parameters based on the previous results.

Provenance and Process Views

Provenance refers to all data about a process, product, or another physical or virtual object that is collected to increase the reproducibility, quality, and understandability [26, 30]. Due to today's heterogeneity of quantum computers with different technologies and characteristics, as well as their high error rates, gathering this data is especially important in the quantum computing domain [46, 88]. Thereby, provenance data can, e.g., help to find the reason for errors occurred or to improve the quantum applications based on passed executions. As the characteristics of quantum computers change during periodic re-calibrations, the provenance data about them must also be retrieved regularly [76]. Further, data about the quantum application, such as the number of iterations or intermediate results for variational algorithms, are important for later analysis [88].

With the increasing complexity of business processes and corresponding workflows, it is challenging to understand, monitor, and analyze them successfully [16, 74]. This complexity even grows when integrating quantum computations into these workflows [47, 91]. Thus, so-called *process views* can be utilized to reduce the complexity and focus on the relevant aspects to accomplish a certain task [9, 69]. A process view is generated by applying specific transformations to a workflow model and is intended to abstract from unnecessary details and make complex workflows easier to understand. In addition to these abstractions, process views can also provide a personalized perspective on the workflow for a specific user [66, 69]. The generation of such personalized process views might include summarizing information, as well as hiding or filtering them.

Problem Statement

By modeling hybrid quantum applications using workflows, one can benefit from the various advantages of workflow technologies. In addition to the advantages presented in the previous sections, also the increased modularity by separating the functionality into independent and reusable services, as well as the understandability due to the graphical notation, have to be taken into account. However, orchestrating the interleaved execution of quantum and classical programs using workflows is inefficient due to the queuing and data transfer times. Instead, hybrid runtimes should be used for the workflow parts comprising such an interleaved execution. Therefore, the first research question (RQ) we are resolving in this work is as follows:

> **RQ 1:** *"How can workflows implementing hybrid algorithms be modeled independently of the runtime to use and automatically be analyzed and rewritten to benefit from hybrid runtimes?"*

We are addressing RQ 1 by splitting it into the following questions tackling the two main challenges:

> **Sub-RQ 1.1:** *"How can workflows be analyzed to detect hybrid loops, which can be executed more efficiently using hybrid runtimes?"*

> **Sub-RQ 1.2:** *"How can programs for hybrid runtimes automatically be generated based on such loops, and workflows be rewritten to use them?"*

However, rewriting the workflow complicates monitoring and analysis, as the modeled and rewritten workflows differ. Therefore, the user must understand the performed changes to properly monitor and analyze the workflow. To overcome this issue, process views can be used. Thus, we are additionally addressing RQ 2:

RQ 2: *"How can process views be utilized to support the user in understanding, monitoring, and analyzing rewritten workflows?"*

Quantum Workflow Analysis and Rewrite

In this section, our method to analyze and rewrite quantum workflows for improved execution using hybrid runtimes is introduced (RQ 1). Furthermore, it supports the monitoring and analysis of rewritten workflows by collecting required provenance data and generating suitable process views (RQ 2). Figure 2 shows an overview of the analysis and rewrite method involving eight steps, which are discussed in the following subsections.

Workflow Modeling

First, the quantum workflow is defined by the user in a manual step. The resulting workflow model comprises all required quantum and classical tasks, as well as the data and control flow between them [20, 51]. Thereby, the QuantME modeling constructs can be used to model the quantum-specific tasks [86]. In the following, we use the BPMN [62] concepts and their graphical notation, as they are widely used and well-known. However, QuantME can also be applied to other imperative workflow languages, such as BPEL [91]. The result of the first step is a workflow model orchestrating the hybrid quantum application, which can comprise one or multiple hybrid algorithms and additional classical tasks.

Candidate Detection

In the second step, candidates for an optimized execution using a hybrid runtime are detected. Such candidates are parts of the workflow orchestrating the interleaved execution of quantum and classical programs (see section "Hybrid Quantum Algorithms & Hybrid Runtimes"). Hence, the quantum workflows are automatically analyzed to identify workflow parts satisfying the following three conditions: (1) they have to contain one or multiple QuantME quantum circuit execution tasks to execute quantum circuits on a quantum computer [91]. If the workflow part does not execute quantum circuits, a hybrid runtime is not needed, and the deployment, e.g., in the cloud is sufficient. (2) In addition

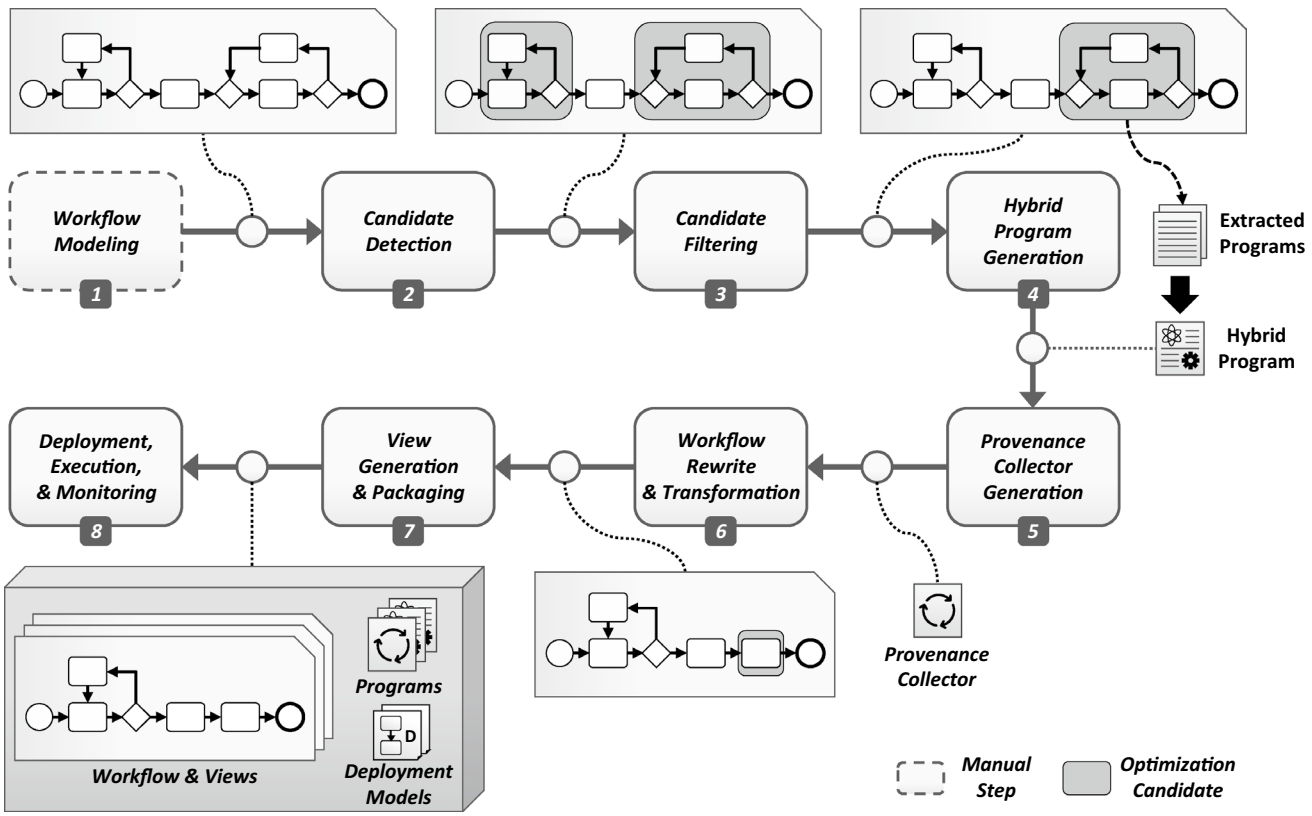

Fig. 2 Analysis and rewrite of quantum workflows to benefit from hybrid runtimes while preserving the workflow provenance

to the execution of quantum circuits, the workflow part must also orchestrate the execution of classical programs or services. This can, e.g., be modeled using service or script tasks in BPMN. Otherwise, batch processing can be used to execute all quantum circuits, reducing the queuing times. Furthermore, the quantum circuits of different quantum circuit execution tasks might also be concatenated into one circuit, e.g., if the quantum computer provides more qubits than required to execute a single quantum circuit. Thus, only the concatenated circuit has to be executed then. However, this can increase the depth and width of the quantum circuit, and hence, a thorough analysis of the quantum circuits and the available quantum computers is needed. (3) Finally, the quantum circuits and classical programs from conditions (1) and (2) have to be executed interleaved multiple times. Otherwise, the queuing and data transfer times do not sum up, and access to the quantum computer through a queue or by reserving a time slice is typically sufficient [83]. The interleaved execution can be modeled in various ways, e.g., by defining a loop in the workflow model or by modeling several tasks in a row. However, the efficiency improvements that can be achieved using hybrid runtimes depend on the number of iterations between quantum and classical programs. Therefore, an open research question for future work is how many iterations are required to benefit from hybrid runtimes, and what other factors, such as the size of the data that has to be transferred between the programs, must be taken into account. In this paper, we restrict the identification of candidates to hybrid loops that can be defined in BPMN, e.g., using two splitting and merging gateways or by directly connecting tasks using conditional sequence flow.

Candidate Filtering

Next, the detected candidates are analyzed in detail and filtered if they are unsuitable for the execution within a hybrid runtime. For this, the properties of the candidates, the characteristics of the hybrid runtimes, and possible restrictions for the supported hybrid programs are considered. As the hybrid runtimes provide different capabilities, the filtering must be performed for each candidate and hybrid runtime combination independently, and a candidate might only be suitable for a subset of the hybrid runtimes. If the filtering results in no suitable hybrid runtime for a candidate, no optimization is possible, and the candidate is not considered in the following steps (see first candidate in Fig. 2). Thereby, various aspects are relevant for the filtering:

(1) Programming languages: Hybrid runtimes usually only support a certain set of programming languages for which they provide the corresponding execution environment with all dependencies. As the hybrid programs are generated from the quantum and classical programs implementing the tasks within the candidate in our approach, the candidate must be filtered if the used languages are not supported. This includes the classical programs, which, e.g., might only be implemented in Python. Furthermore, the libraries that can be used in the programs can be restricted too if the hybrid runtime does not allow installing arbitrary libraries and only provides a preinstalled set. On the other hand, it also applies to quantum programs for which only OpenQASM or Quil might be supported.

(2) Activity and task types: Additionally, depending on the used workflow language, some of the available activity and task types can not be properly mapped into a hybrid program. Therefore, they are not allowed within optimization candidates. For example, if the hybrid runtime does not provide the capability for transactional processing, transactions are also not admitted in candidates, as the rewrite would change the semantics otherwise. Furthermore, human tasks often lead to long delays. However, this prevents fast iterations between classical and quantum programs and contradicts the basic idea of hybrid runtimes. Finally, for some cases, the filtering has to be performed recursively. For example, this applies to sub-processes or modeling constructs such as the BPMN call activity invoking globally defined tasks or workflows. However, the filtering can only be applied recursively, if the corresponding definitions are accessible.

(3) Events: Also, some of the events that can be defined in various workflow languages can not be represented in hybrid programs. One reason for this is impractical delays incurred by receiving events, such as catching message or signal events, or other blocking events, e.g., timer events. Further types of events lead to technical difficulties when trying to realize them within hybrid programs. For example, to trigger the compensation of parts of the workflow, if a throwing compensation event is part of the hybrid program, it must be mapped to an API call at the workflow engine. Thus, this has to be supported by the hybrid runtime, and the corresponding hybrid program must be generated accordingly if the candidate comprises such an event.

(4) Gateways: Similar to events, gateways can be difficult to realize within hybrid runtimes if no workflow engine is used internally. For example, if a gateway relies on external events that have to be handled by the hybrid program then. Another example are gateways controlling the sequence flow based on global semantics, such as the inclusive gateway in BPMN, as evaluating their conditions is complicated in hybrid programs.

(5) Quantum computer provider: Finally, the concrete quantum computer to use or the corresponding provider can restrict the set of hybrid runtimes that are suitable for optimizing a candidate. Thereby, the quantum circuit execution tasks within the candidates can be configured to execute the quantum circuits using a specific quantum computer

or provider if necessary. However, then the target hybrid runtime must provide access to this quantum computer or provider.

Independent of the characteristics and capabilities of the different available hybrid runtimes, the source code of the quantum and classical programs implementing the tasks within the candidates must be available. For script tasks, the source code is often packaged together with the workflow model, either as a separate script in an archive or inline within the workflow model definition [62]. Additionally, there are approaches to attach deployment models for services that are needed directly to the corresponding activities and package them with the workflow model [90]. These deployment models contain the required code, which can then be retrieved when generating the hybrid program. In contrast, the source code might not be available for external services, e.g., available through a service registry [18]. Then, the generation of a hybrid program is not possible.

If the filtering step results in multiple suitable hybrid runtimes for a candidate, the user decides which hybrid runtime to use. However, as part of future work, this selection can be automated based on non-functional requirements, such as costs or trust in the provider.

Hybrid Program Generation

In the fourth step, for each candidate, an equivalent hybrid program is generated, which can be executed on the selected hybrid runtime from the previous step. First, the quantum and classical programs of the candidate are extracted (see Fig. 2) [50]. Additionally, new code snippets are created to implement the functionality of the other modeling constructs in addition to the tasks within the candidate, such as events or gateways. For this, our approach relies on templates for the various supported modeling constructs [63]. These templates provide the generic logic for the corresponding modeling construct and can be instrumented with the variable parts, e.g., the condition for a gateway. To successfully invoke the hybrid program by the workflow, the input and output parameters must be determined. This is achieved by analyzing the workflow and collecting the input parameters of all modeling constructs within the candidate. These input parameters are filtered if they are the output of another modeling construct of the candidate, as the data can be passed within the hybrid program then. In the same way, the output parameters of the hybrid program are retrieved by collecting the output parameters within the candidate and filtering them if they are not used outside the candidate. Next, all code fragments, i.e., the extracted programs and the generated code snippets, are combined into the hybrid program based on the sequence flow defined in the candidate. Furthermore, the equality of the data flow must also be ensured when merging the code fragments. This is realized by introducing a set of variables in the hybrid program based on the input and output parameter analysis, as well as the data flow within the candidate, e.g., specified using data objects in BPMN.

Provenance Collector Generation

Next, a *provenance collector* is generated to retrieve provenance data about the hybrid runtimes and the hybrid programs when executing them [88]. This data can later be used to monitor and analyze the workflow execution using process views (see section "Provenance & Process Views"). The provenance collector is specific to the hybrid runtime selected for a certain candidate, as the APIs of the hybrid runtimes differ or the collectors must be implemented in various programming languages [4, 34]. Thus, if different hybrid runtimes were selected for the candidates of the workflow model in step 3, multiple provenance collectors must be generated. The functionality of the provenance collector and how it retrieves provenance data from hybrid runtimes are discussed in section "Provenance Retrieval from Hybrid Runtimes".

Workflow Rewrite and Transformation

In the sixth step, the workflow is rewritten to invoke the generated hybrid programs instead of orchestrating the whole workflow parts of the candidates. The invocation of a hybrid program can be realized by a new service task, which starts the execution of the hybrid program and retrieves the results through the API of the hybrid runtime. Therefore, all workflow parts corresponding to candidates are each replaced by a single service task. Thereby, in- and outgoing sequence and data flow are redirected from the workflow parts to the new service tasks. Furthermore, also boundary events of the replaced tasks are attached to the service task if their sequence flow leaves the candidate. In contrast, boundary events redirecting the sequence flow within the candidate are handled in the hybrid program (see section "Hybrid Program Generation"). However, only some boundary events might be supported. For example, an interrupting timer boundary event requires that the hybrid program is canceled once the timer expires. Hence, the cancellation must be enabled by the hybrid runtime. If one of the events can not be realized, the workflow rewrite for this candidate must be aborted. Otherwise, the semantics of the quantum workflow would be changed.

In addition to the rewrite, the quantum workflows are transformed in this step. This means all remaining QuantME modeling constructs are replaced by reusable workflow fragments implementing the required functionality [21, 91]. The fragments are created by quantum experts and, e.g., comprise the logic to apply a certain encoding during data preparation or to execute a quantum circuit at a specific provider.

After this transformation, the workflow only contains native modeling constructs of the used workflow language. Hence, the portability between workflow engines is retained.

View Generation and Packaging

In the next step, process views are generated for the monitoring and analysis of the quantum workflow. Our main use case is to visualize the original workflow before rewriting it to increase understandability. However, similar to the rewriting, also the transformation leads to a difference between the modeled and executed workflow, as the user defines the QuantME modeling constructs, which are replaced by corresponding workflow fragments before execution. Therefore, providing a process view before and after this transformation can ease monitoring and analysis. All details about process views for quantum workflows are discussed in section "Process Views for Quantum Workflows".

The generated hybrid programs and provenance collectors must be deployed to the hybrid runtimes before the workflow can be executed. However, a manual deployment would be time-consuming and error-prone [10]. To automate this deployment, deployment models are generated and attached to the activities within the workflow [90]. The programs, deployment models, and the workflow model with all views are then packaged in a self-contained archive (see Fig. 2) [47]. Hence, only this archive must be transferred into the target environment where the workflow is executed.

Deployment, Execution, and Monitoring

Finally, in the last step, the execution environment for the quantum workflow is set up. This includes deploying all services and programs that are not always on and need to be provisioned before executing the workflow [86]. For example, the classical services can be deployed in the cloud or using a local computer. Additionally, the hybrid programs and provenance collectors are provisioned using the deployment models from the previous step. Thereby, various deployment systems can be utilized, e.g., *Kubernetes* [15], *Terraform* [28], or the *OpenTOSCA Container* [8]. To successfully invoke the deployed services and programs, they are bound to the workflow after the deployment [90]. This can, e.g., be achieved by adding the information about their endpoints to the activities of the workflow model [51].

When the binding is completed, the updated workflow model is uploaded to a workflow engine. Then, it can be instantiated by passing the required input data, e.g., the tokens to access the hybrid runtimes. When a service task invoking a hybrid program is reached, the provenance collector starts gathering provenance data about the hybrid runtime and the current execution (see section "Provenance Retrieval from Hybrid Runtimes"). This provenance data is then retrieved by the workflow engine and used to instrument the process views, e.g., with the current state of the execution and possible intermediate results.

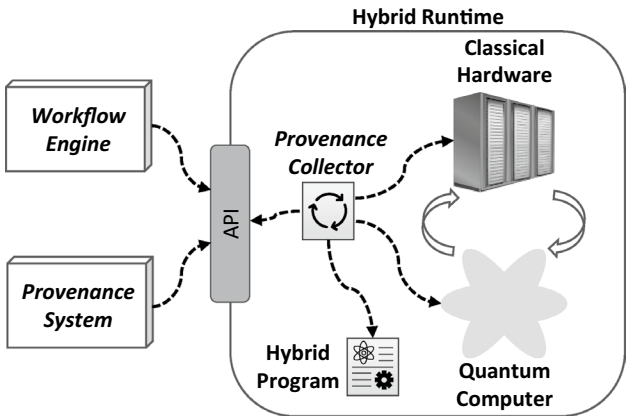

Fig. 3 Retrieval of provenance data about hybrid runtimes

Monitoring and Analysis of Rewritten Quantum Workflows

In the following, we introduce our approach to retrieve provenance data about hybrid runtimes and hybrid programs. Further, it is shown how this data can be used to instrument process views for quantum workflows.

Provenance Retrieval from Hybrid Runtimes

To monitor and analyze hybrid programs executed in a hybrid runtime and to support the usage of process views for quantum workflows, detailed provenance data must be collected. The data that should be gathered can be divided into two categories: (1) first, provenance data about the hybrid runtime itself, i.e., the quantum computer and classical hardware. This comprises, e.g., the number of qubits, their decoherence times, gate errors, or the topology of the quantum computer, as well as the CPU or RAM of the classical hardware [88]. Such data is usually made available by the provider via an API and can be retrieved by a provenance system or workflow engine (see Fig. 3) [46]. (2) Furthermore, data about the running hybrid program must be collected, e.g., the number of the current iteration with intermediate results for a variational algorithm. However, provenance data from this category are typically not directly made available by the provider via the API [4, 33]. Thus, it is gathered by the generated provenance collector from step 5 of our method (see section "Provenance Collector Generation"). The provenance collector is deployed together with the hybrid program and runs on the classical hardware inside the hybrid runtime, as depicted in Fig. 3.

Depending on the hybrid runtime, it can be a separate program or also be integrated into the hybrid program with statements to collect the required data. This is needed if the hybrid runtime only allows to deploy a single program, such as Qiskit Runtime [33]. The collected provenance data can be returned regularly via the API if it supports providing intermediate results, e.g., to enable live monitoring, or all data is returned with the final result.

Process Views for Quantum Workflows

Process views for quantum workflows can be used to increase their understandability and ease the monitoring and analysis after rewriting or transforming the workflow. Figure 4 shows an excerpt of the workflow model after applying the analysis and rewrite method, as well as a corresponding process view, based on the initial example presented in Fig. 1. On the left, the rewritten workflow model is depicted that is executed in the workflow engine, comprising only a single service task to invoke a hybrid program. In contrast, the right side shows a process view of the workflow before rewriting, orchestrating a hybrid loop with two gateways and three tasks. It can be seen that the *process token*, indicating the currently active task, is located at the service task on the left. This means that no further information about the current state of the hybrid program is available. Visualizing the process view, the token moves between the different tasks of the hybrid loop, enabling the user to monitor the progress. However, this requires that intermediate results are returned from the provenance collector as described in the previous section. Otherwise, the process view can only be used for analysis after the execution. In the same way, the set of variables that can be displayed differ. While in the executed workflow model only the input parameters and the output after finishing the execution are available, the process view provides further information, e.g., the current iteration, the counts from the last ansatz execution, or the current costs. In addition to more detailed process views for rewritten workflows, also abstractions for the workflow transformation can be applied (see section "Workflow Rewrite & Transformation"). Thereby, abstract QuantME tasks are replaced by workflow fragments implementing their logic, which can comprise multiple tasks. Thus, a process view can abstract from these technical details for some users and visualize the workflow with the QuantME tasks.

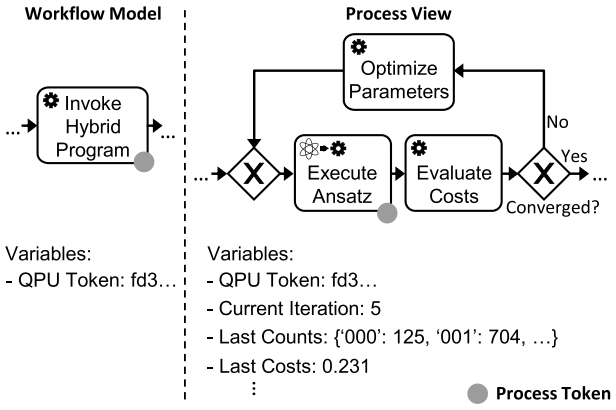

Fig. 4 Process view for a rewritten quantum workflow

Prototypical Validation

In this section, we present the system architecture of the MODULO framework supporting the concepts introduced in sections "Quantum Workflow Analysis and Rewrite" and "Monitoring & Analysis of Rewritten Quantum Workflows". Furthermore, we discuss our corresponding prototypical implementation.

System Architecture

In Fig. 5, an overview of the system architecture of the extended *MODULO framework* [86] is given. The MODULO framework enables to model, transform, deploy, and execute quantum workflows. In this paper, we extend it to also support the analysis and rewrite method, as well as the monitoring and analysis using process views. Thereby, already existing and unchanged components are light, expanded components are gray, and newly introduced components are dark.

First, MODULO comprises the *QuantME Transformation Framework*, as depicted in the middle of Fig. 5. The QuantME Transformation Framework is a graphical BPMN workflow modeler based on the Camunda modeler [12] that was extended to support QuantME. For the candidate detection and filtering, a new plugin-based *Workflow Analyzer* was added. It uses a plugin for each supported hybrid runtime, which specifies the restrictions and filtering rules for the categories described in section "Candidate Filtering". Next, the *Workflow Rewriter* component is responsible for rewriting the quantum workflows based on the analysis and filtering. For this, it extracts the quantum and classical programs for each candidate and sends them to the *Hybrid Runtime Handler* supporting the selected hybrid runtime.

The Hybrid Runtime Handlers are depicted on the left, and currently, two handlers are supported: (1) the *Qiskit Runtime Handler* and (2) *Amazon Braket Handler*. Both comprise the same components, and further handlers can be added once new hybrid runtimes are emerging. The *Input Parser* provides a plugin for each supported programming language, which analyzes the quantum and classical programs and extracts the code that must be merged into the hybrid program. Furthermore, the handlers provide a

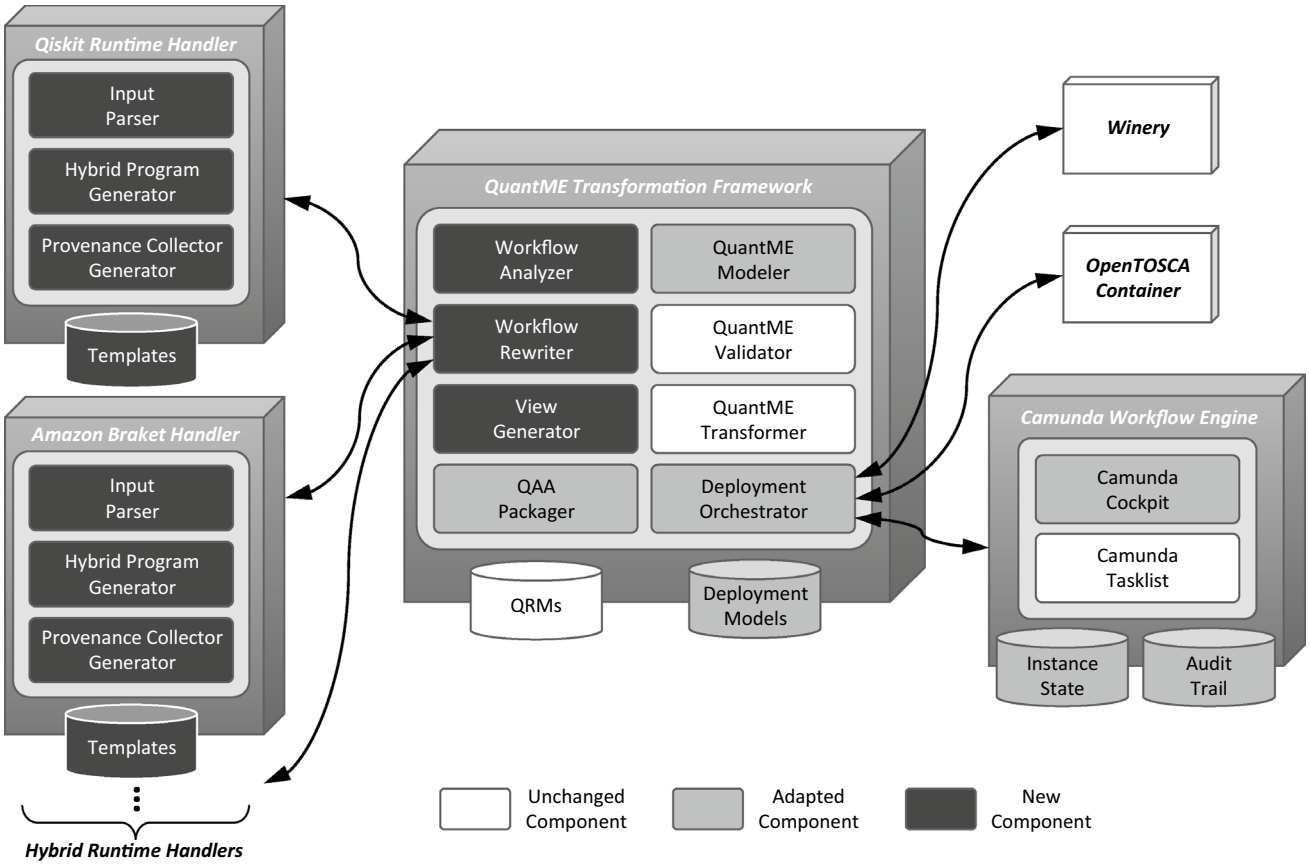

Fig. 5 System architecture of the extended MODULO framework supporting the analysis and rewrite method

repository with *Templates*, i.e., code snippets that are instrumented to implement the functionality of events and gateways within a candidate. Based on the code snippets and the code from the programs, the hybrid program is created by the *Hybrid Program Generator* while ensuring the correct sequence and data flow. The *Provenance Collector Generator* is capable of generating provenance collectors for the specific hybrid runtime either as separate programs or by including the statements to gather data into the hybrid program. Finally, the hybrid program and the provenance collector are sent back to the Workflow Rewriter.

Further, the QuantME Transformation Framework is extended by a *View Generator* to create different process views during workflow rewrite and transformation, which can be selected by the user for monitoring and analysis. To package all required artifacts in a single self-contained archive, the *Quantum Application Archive (QAA) Packager* was adapted to include the generated process views in addition to, e.g., deployment models or the workflow model. The *QuantME Modeler* enables graphically modeling BPMN workflows and supports QuantME. It was extended to assist the user during workflow analysis and rewrite, e.g., visualizing candidates and enabling the selection of a hybrid runtime if multiple are suitable. Two unchanged components are the *QuantME Validator* to detect errors in workflows and the *QuantME Transformer* to transform workflow models comprising QuantME modeling constructs to native workflow models. For this transformation, a repository with *QuantME Replacement Models (QRMs)* is used, defining how QuantME modeling constructs are replaced by reusable workflow fragments. All deployment-related functionality is implemented in the *Deployment Orchestrator*, which uses *Winery* [40] to store and retrieve deployment models, and the *OpenTOSCA Container* [8] to automate the deployment.

The Deployment Orchestrator also manages the upload of the workflow model and the process view definitions to the *Camunda Workflow Engine* [13], a state-of-the-art BPMN workflow engine. Thereby, the *Camunda Cockpit* component visualizes running and finished workflow instances, e.g., showing the process token or the current variable values. It uses the *Instance State* database for running workflow instances, and after termination, the data is moved to the *Audit Trail*. It was extended to visualize process views and to instrument them with up-to-date provenance data. The

Camunda Tasklist provides a graphical interface for the user to work on human tasks and is used unchanged.

Prototypical Implementation

The QuantME Transformation Framework is realized as a standalone desktop application using the *Electron framework*. It consists of a graphical user interface implemented in JavaScript and a Node.js backend. Thereby, the user interface was extended to guide the user through all steps of the analysis and rewrite method. On the other hand, the workflow analysis and rewrite, as well as the communication with the new external components was implemented in the backend.

Further, we prototypically implemented two Hybrid Runtime Handlers, namely the Qiskit Runtime Handler and the Amazon Braket Handler. Both are based on Python as the Braket and the Qiskit SDK are implemented in Python. Thus, the code analysis is eased, and also the required code snippets for gateways or events are available as Python files. For the execution of generated hybrid programs, Qiskit Runtime [33] and Amazon Braket Hybrid Jobs [4] are used. However, the hybrid runtimes have different requirements for hybrid programs, which must be reflected in the corresponding handler. For example, Qiskit Runtime currently only supports a single file as a hybrid program. Hence, all required code must be merged into this file, also comprising the provenance collector. Additionally, only a limited set of dependencies is available in the execution environment, and if other dependencies are needed, e.g., another optimizer, the hybrid program can not be executed. Therefore, the generation is aborted in this case. However, these are restrictions in the current state and are planned to be resolved when Qiskit Runtime is released with full functionality after beta mode [35]. In contrast, Amazon Braket Hybrid Jobs enables custom dependencies, but they have to be provided in a Docker image for the execution. This means a Docker image can be created and uploaded to the registry, where Amazon Braket Hybrid Jobs retrieves the images.

Finally, the Camunda engine provides plugin points in its graphical user interface, which was used to add a JavaScript plugin to visualize process views and the provenance data retrieval to instrument them. The prototypical implementations of all components are publicly available as open-source projects on Github [79].

Case Study

In this section, we discuss an exemplary quantum workflow and showcase how it can be analyzed, rewritten, and monitored to validate the practical feasibility of our concepts and the corresponding prototype. The workflow is taken from the quantum humanities domain and applies clustering and classification to costume data [5, 6]. Figure 6 shows the workflow model on the top.

The workflow starts with a message start event, which receives request messages with the required input data. For the sake of simplicity, we assume that the input data is already pre-processed and all classical tasks are already performed. This comprises, e.g., transforming categorical data to numerical data or reducing the number of dimensions [5, 6]. However, these tasks can also be integrated into the workflow, e.g., using additional service tasks at

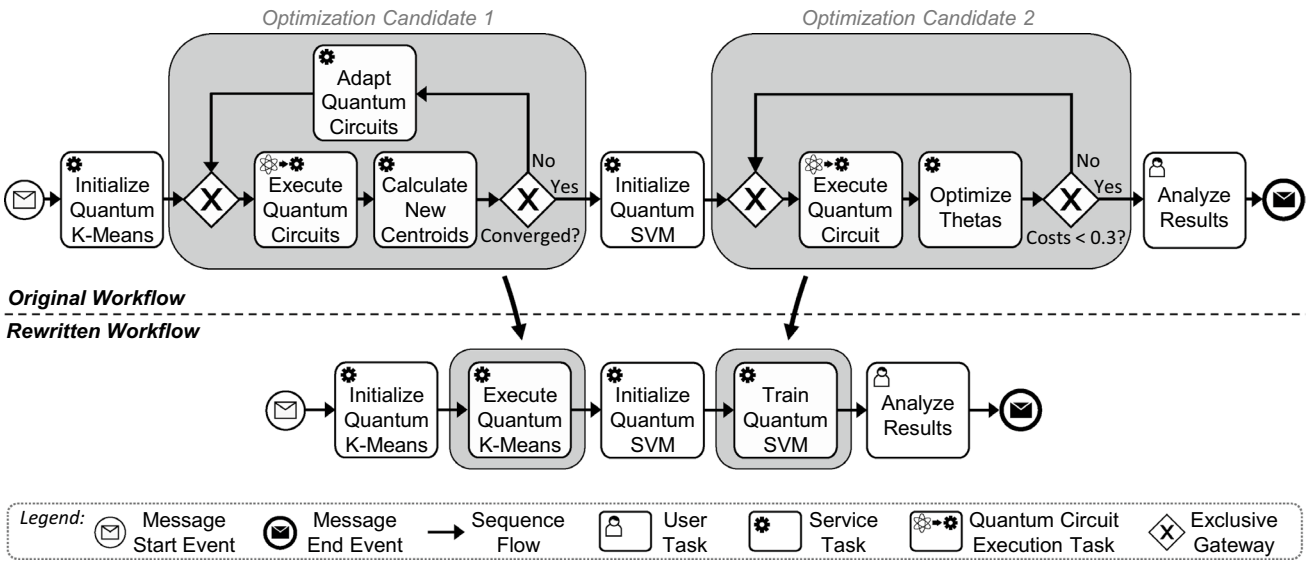

Fig. 6 An exemplary quantum workflow comprising two hybrid loops which can be optimized using a hybrid runtime [85]

Table 1 Runtime evaluation of the analysis and rewrite method for different workflow models and hybrid runtime providers

Workflow ID	Provider	Number of activities	Number of candidates	Avg. activities per candidate	Candidate detection (s)	Program generation and rewrite (s)
1	IBM	8	2	2.5	0.31	107.48
1	AWS	8	2	2.5	0.32	108.13
2	IBM	50	2	2.5	0.45	108.16
3	AWS	50	4	2.5	0.48	216.42
4	IBM	50	6	2.5	0.75	324.97
5	AWS	50	2	5	0.43	297.46
6	IBM	50	4	5	0.58	612.05
7	AWS	50	6	5	0.70	898.83

Thereby, workflow 1 was evaluated for both providers to compare the performance of the different Hybrid Runtime Handlers

the beginning of the workflow. After receiving the pre-processed input data, clustering is performed using the *quantum k-means algorithm* [39], consisting of multiple tasks that need to be orchestrated. First, the quantum algorithm is initialized by determining random initial centroids for the clustering and based on these centroids corresponding quantum circuits are generated. Then, a hybrid loop is entered using the quantum circuit execution task to execute the generated quantum circuits. The measurement results are used to calculate the new centroids in a classical service task. Next, it is checked if the clustering converged, i.e., if the difference between the old and new centroids is smaller than a given threshold. If this is not the case, the next iteration is entered by adapting the quantum circuits to the new centroids.

When the clustering converges, the resulting clusters are utilized to train a classifier using a variational quantum *Support Vector Machine (SVM)* [29]. The algorithm is based on an ansatz, which is generated together with an initial parameterization in the first service task. Similar to the clustering, it then enters a hybrid loop executing the ansatz and optimizing the parameters in each iteration based on the measurement results. The parameters are also used to evaluate a cost function, and if the costs are higher than the threshold of 0.3, the next iteration is entered. Otherwise, the last user task provides details about the performed clustering and the trained classifier, which can be analyzed. Finally, the result is also sent back to the initiator of the workflow execution by the message end event.

By applying the analysis and rewrite method to the workflow, two optimization candidates satisfying all requirements are detected, which are surrounded by gray boxes in Fig. 6. For the candidates it is verified if they can be optimized using one of the supported hybrid runtimes, i.e., Qiskit Runtime and AWS Braket Hybrid Jobs. However, for our case study, we continue by utilizing Qiskit Runtime for the optimization. The candidates do not contain invalid modeling constructs, such as unsupported events (see section "Hybrid Program Generation"). Thus, they are not filtered, and rewriting can be performed for both candidates. At the bottom of Fig. 6, the resulting rewritten workflow is depicted. The two service tasks initializing the hybrid algorithms and the user tasks are unchanged. In contrast, the optimization candidates of the original workflow are both replaced by a service task invoking the generated hybrid program at Qiskit Runtime. During execution, the workflow at the bottom is conducted. However, using the provenance and process view functionalities, users can also visualize the original workflow showing the current position of the process token and variable values in more detail.

The discussed use case, together with a detailed description of how to set up and use the framework, can be found on Github [80]. Furthermore, a corresponding demonstration video is available on YouTube [77].

Evaluation

In the following, we discuss the evaluation of the runtime when applying our method, analyzing and rewriting workflows with different sizes. Additionally, we compare the execution times of various workflows without using our method to the execution times after rewriting and using hybrid runtimes from AWS and IBM. Finally, we evaluate how the collection of provenance data impacts the execution times of hybrid programs.

Runtime of the Analysis and Rewrite Method

First, we evaluate the performance of our analysis and rewrite method to validate its applicability to various workflow models in a reasonable time frame. For this, we modeled seven differently sized workflow models, as summarized in Table 1. Additionally, the evaluation was done for IBM and AWS, i.e., using the Qiskit Runtime Handler and the Amazon Braket Handler for the generation of hybrid programs. For workflow models, there exist different complexity

measures [44]. However, we focus on three attributes characterizing the workflows, directly influencing the execution time of the analysis and rewrite method: (1) the number of activities within the workflow model enlarges the search space for the detection of candidates. Thus, a higher number of activities can increase the required time. (2) Further, the number of candidates that can be detected must be considered. As the number of hybrid programs to generate corresponds to the number of candidates, this strongly impacts the execution time. (3) Finally, the average number of activities per candidate is important. For each activity of a candidate, the corresponding program must be retrieved, analyzed, and merged into the hybrid program leading to increased generation times. In addition to these attributes characterizing the workflows, the language and length of the various programs, i.e., their lines of code, have an impact on the hybrid program generation. Hence, all programs for the workflows of our evaluation are implemented in Python and have a similar length of around 200 lines of code.

Table 1 summarizes the required time to detect the candidates in the workflow, as well as the time to generate corresponding hybrid programs and rewrite the workflow accordingly for each of the workflow models. As the first workflow model for the evaluation, we utilize our case study introduced in section "Case Study". The workflow consists of 8 activities and 2 candidates, comprising 3 and 2 activities. Thus, the average number of activities per candidate is 2.5. This workflow model was evaluated for both providers, i.e., IBM and AWS, to determine if this leads to differences in the program generation and rewriting times. For the candidate detection, the required time is almost equal, as the analysis of the workflow is independent of the Hybrid Runtime Handler. However, also the program generation and rewriting times are similar and not influenced by the selected provider. Thus, we evaluated only one of the two providers for the remaining workflows.

Workflow 2 extends the case study by adding further classical activities which are not related to the candidates. Thereby, the candidate detection takes slightly longer due to the enlarged search space of the new activities. As the number and size of the candidates are the same, the program generation and workflow rewrite times are almost equal to workflow 1. Further, the time for the candidate detection is negligible compared to the program generation and rewriting. Hence, the other workflows do not increase the overall number of activities but add additional candidates or adapt their size to specifically evaluate the program generation and rewrite. Workflows 3 and 4 each add two candidates while retaining the average size of the candidates. The corresponding program generation and rewrite times grow linearly with the number of candidates, as they are processed sequentially in our current prototype. However, this can be parallelized in the future, e.g., by utilizing multiple instances of the Hybrid Program Handlers to handle different candidates of the workflow.

Finally, the average number of activities per candidate was doubled from 2.5 to 5 for workflows 5 to 7. Thereby, between workflows 5 to 7, the program generation and rewriting time again increases linearly. In contrast, the doubled average activities per candidate lead to a growth of approximately factor 2.8 compared to workflows 2 to 4. This is caused by the increased effort to determine the interface of the hybrid program, merge the different code snippets, and ensure the correct sequence and data flow between them.

In summary, the recorded times to apply the analysis and rewrite method to the evaluated workflow models is rather short compared to the runtime of large workflows. These can often take multiple hours, days, or even years [51]. Furthermore, the evaluation of the workflow runtime before and after the rewrite presented in the following section shows that the efficiency improvement already exceeds the required time to apply the method when executing the workflows once. However, after performing the analysis and rewrite method, the workflow can be executed multiple times.

All measurements were performed on a computer running Windows 11 64-bits with an Intel(R) Core(TM) i7-8565U CPU @ 1.80 GHz processor and 40 GB of RAM. Thereby, the median based on 100 measurements for each workflow is calculated. The collected raw data of all performed measurements is available on Github [78].

Workflow Runtime Comparison

Next, we compare the workflow runtime for a subset of the workflow models presented in the previous section before and after rewriting them. First, four workflow models are executed using quantum computers from IBM and Qiskit Runtime for the rewritten workflows. Second, another workflow model is evaluated for AWS, i.e., utilizing a quantum computer from Rigetti and the AWS Braket Hybrid Jobs functionality. Please note that a direct comparison of the two hybrid runtimes under the same assumptions and preconditions is not possible for our use cases. The reason for this are the different quantum computers that can be used through the hybrid runtimes. They provide a varying number of qubits, different connectivities between them, and diverse error rates. However, these characteristics impact the measurement results of the quantum circuits, which influences the number of iterations within the hybrid loops of the workflows. Consequently, this also changes the overall execution times of the workflows. Furthermore, the gate times differ for the quantum computers, i.e., the quantum circuits themselves have also diverging execution times. Thus, we compared both hybrid runtimes to the same quantum computer without this functionality, showcasing that our approach leads to shorter execution times independent of the provider

and hybrid runtime. Another important factor influencing the number of quantum circuits to execute and accordingly the workflow execution times, is the size of the input data. Hence, exactly the same input data was used for the original and rewritten workflows. Finally, the data presented in the following box plots was collected from ten executions of the original and rewritten workflows.

Workflow Runtime Comparison for IBM

The evaluation of the workflow runtimes directly using quantum computers from IBM and comparing them to the usage of Qiskit Runtime includes four workflow models for which the execution times are depicted in Fig. 7. Both the quantum circuits of the original workflows and the hybrid programs of the rewritten workflows were executed on the *ibmq_ehningen* quantum computer. The quantum computer is an *IBM Q System One*, comprising 27 qubits. Furthermore, an exclusive reservation of the quantum computer was used for the evaluation. Therefore, quantum circuits submitted over the queue did not have to wait for other jobs to complete, which is the best case to execute the original workflows. Thus, the accomplished speed-up is solely based on the optimized execution of Qiskit Runtime, and other jobs in the queue can further increase it. For workflow 1, the median execution time using *Quantum Computing as a Service (QCaaS)*, i.e., directly executing quantum circuits over a queue, was 569 s. In contrast, using Qiskit Runtime the median execution time decreases to 304 s, i.e., roughly a speed-up of factor 1.9 was achieved. Workflow 2 has a median execution of 699 s when using QCaaS and 425 s utilizing Qiskit Runtime. While the absolute runtime improvement is similar to workflow 1, the relative speed-up is smaller as workflow 2 only adds additional classical tasks which can not benefit from the hybrid runtime. Workflow 4 comprises more candidates, and thus, more hybrid programs to execute. While the original workflow has a median execution time of 1454 s, the rewritten workflow only requires 749 s, again providing a speed-up of about factor 1.9. Finally, for workflow 7, the execution time changed from 1768 to 748 s when rewriting the workflow. The reason for the higher speed-up of factor 2.4 is the increased number of activities among the candidates, which can benefit from hybrid runtimes.

Workflow Runtime Comparison for AWS

For AWS, one workflow model was evaluated for which Fig. 8 visualizes the execution times. AWS provides access to quantum computers from different quantum hardware providers, such as IonQ or Rigetti. We selected the *Rigetti Aspen-11* for our experiments, which provides 38 qubits. However, for AWS no exclusive reservation was possible, and the quantum circuits executed for the original workflow had to be submitted to the queue. In contrast, using AWS Braket Hybrid Jobs preferred access to the quantum computer is provided once the hybrid program starts [4]. On the

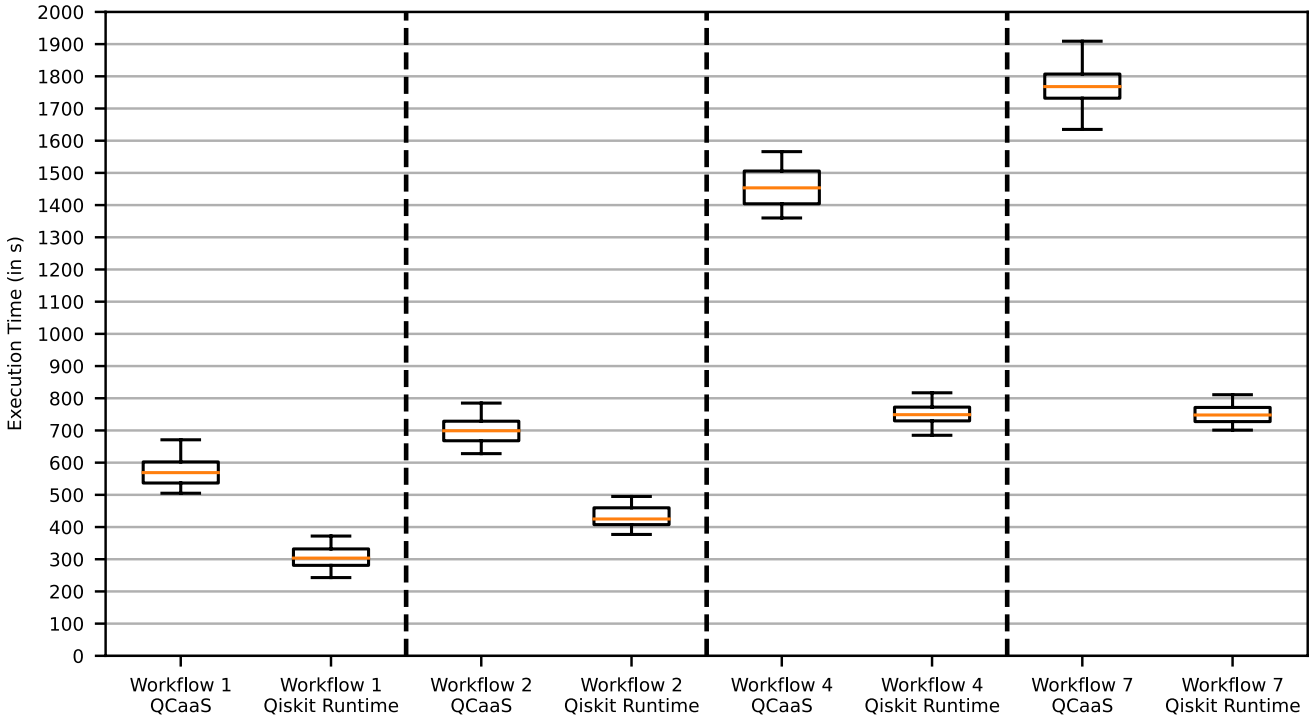

Fig. 7 Workflow execution times for different example workflow models using IBM with and without Qiskit Runtime

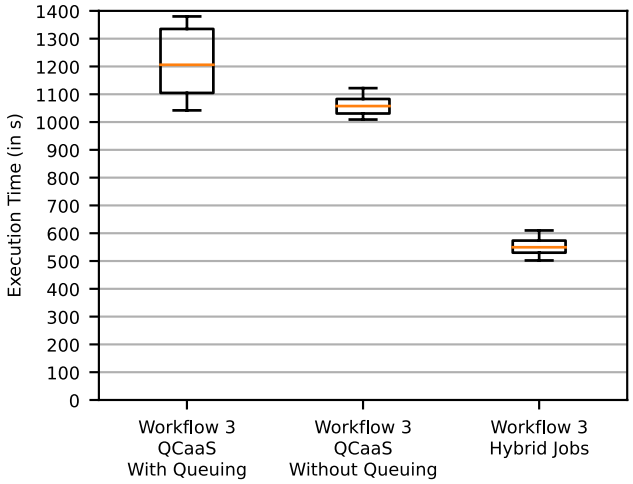

Fig. 8 Workflow execution times using AWS with and without Hybrid Jobs

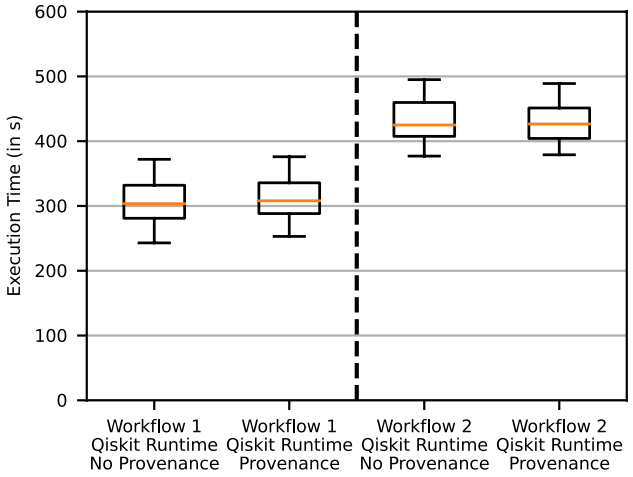

Fig. 9 Workflow execution times for Qiskit Runtime with and without provenance collection

left of Fig. 8 the execution times with queuing are shown, having a median of 1206 s. We collected the queuing times for each circuit execution and subtracted them from the overall execution times, leading to the data depicted in the middle. The median decreased to 1058 s, and the variance is way smaller, as the queuing times strongly vary. Finally, on the right, the execution times for the rewritten workflow using AWS Braket Hybrid Jobs are shown. The median execution time is 550 s, providing a speed-up of about factor 1.9 compared to the original workflow without queuing and factor 2.2 including queuing.

Runtime Comparison with Provenance Collection

As the last part of the evaluation, the workflow runtime using Qiskit Runtime is compared when running the hybrid program with and without provenance collection. This can influence the performance as additional statements are executed within the hybrid program or as part of a separate provenance collector running on the classical hardware of the hybrid runtime. Figure 9 shows the execution times for workflow 1 and 2 using Qiskit Runtime with and without collecting provenance data about the hybrid program execution. Thereby, the data from Fig. 7 was used for the execution without collecting provenance data. For workflow 1, the median execution time increased by 4 s from 304 to 308 s. Similarly, the execution time for workflow 2 is 425 s without and 427 s with provenance collection. However, the differences for both workflows are within the variance of the measurements, and no serious increase is to be noted. Hence, the provenance collection can be performed without impacting the overall performance, enabling to use process views for monitoring workflows.

Discussion

In this section, possible improvements in the detection of suitable candidates and extensions with new kinds of hybrid runtimes are discussed. Additionally, the automated selection of a suitable quantum computer before rewriting the quantum workflows is explored.

Current hybrid runtimes incorporate quantum computers with co-located classical hardware, whereby typically a containerized execution environment is provided to execute single scripts. However, in the future, different sets of prebuild runtimes are expected to evolve, as, e.g., announced by IBM in their quantum computing roadmap [32]. For example, such runtimes can incorporate a workflow engine or an HPC, which are co-located to the quantum computers. Our approach can be extended to support these runtimes by adding corresponding detection and filtering rules. Furthermore, the generation of the software artifacts for the runtimes must then be changed, e.g., using sub-workflows to execute within a runtime comprising a workflow engine.

For the candidate detection and filtering, only the structure of the workflows, i.e., the tasks, events, gateways, and the corresponding quantum and classical programs, as well as the characteristics of the hybrid runtimes, are considered at the moment. However, including non-functional requirements in the detection and filtering, as well as the selection of a concrete hybrid runtime to use, can improve the resulting rewritten workflow [19, 67, 73]. Thus, non-functional requirements or policies can be attached to tasks, e.g., that it comprises confidential data that is not allowed to be transferred to specific providers. Another example would be annotating if a certain task increases or decreases the data size. To improve the efficiency, tasks decreasing the size should be performed

before transmitting the data between the hybrid runtime and the workflow engine [93].

The quantum circuit execution tasks allow defining a provider or even a specific quantum computer to use, which also restricts the set of possible hybrid runtimes for the rewrite. If not needed, e.g., due to the trust in the provider, this should be avoided to increase the number of possibilities during detection and filtering. Instead, the quantum computer for the execution of quantum circuits within quantum workflows can also be automatically selected based on different strategies [87].

Related Work

The adaptation of workflow models during design time or dynamically during runtime is discussed by different research works, which are presented in the following.

Bucchiarone et al. [11] introduce an approach to model workflows abstractly, where the tasks are annotated with goals, preconditions, and effects. These tasks are then dynamically replaced with fine-grained workflow fragments implementing the required functionality. Mundbrod et al. [59] follow a similar approach, adapting workflows by injecting workflow fragments depending on the current context, e.g., the available resources or the load on a component. Also, Képes et al. [38] adapt workflow models by dynamically selecting and executing suitable workflow fragments based on the current situation. In contrast to these approaches transforming from an abstract workflow model to a more fine-grained one, our approach requires the opposite direction summarizing the workflow part of a candidate within a single service task. Furthermore, it does not rely on predefined workflow fragments, but the required hybrid programs are automatically generated.

Cohen-Boulakia et al. [16] discuss how to rewrite scientific workflows to satisfy series-parallel structures enabling efficient processing of provenance data. However, they only adapt the workflow by rearranging the tasks and do not generate new programs or services. Wang et al. [84] introduce a scheduling procedure for scientific workflows based on trust in different service or hardware providers. Depending on the result, they adapt the sequence flow of the workflow. Additionally, various approaches are generating new workflow models or adapting existing ones using the data collected during workflow execution in the audit trail of the workflow engine by applying process mining techniques [2, 81, 82].

The adaptation or transformation of workflows during runtime is also directly implemented in some workflow engines, so-called *adaptive workflow engines*. For example, *AristaFlow* [68] enables the adaptation of deployed workflow models, as well as the automated migration of workflow instances to a new version of the workflow model. *AgentWork* [58] is another example of an adaptive workflow engine. It automatically adapts workflows in exception cases by adding or removing activities instead of requiring to model all possible alternative sequence flows in the workflow. However, to support our approach, the workflow engines must be extended, reducing the portability of the workflows. In contrast, the presented approach transforms the workflow models to native workflow models before deployment and execution. Thus, the portability is retained, and only the visualization of the process views is specific to the used workflow engine, which is an optional feature if needed by the user. In previous work [87], we presented an approach to select a suitable quantum computer for the execution of quantum circuits during runtime of quantum workflows. Thereby, the selection is based on characteristics of the quantum circuits, such as their depth, which are based on the input data. Thus, it can only be performed during runtime, and the workflow is adapted to invoke the quantum circuits using the selected quantum computer. In future work, we plan to incorporate this approach with our analysis and rewrite method to dynamically rewrite the workflow for an automatically selected quantum computer and hybrid runtime. However, the generation of hybrid programs during workflow runtime can lead to longer delays, thus, the user can decide if the rewrite during design or runtime is more suitable for his use case.

Views are used in various application domains to introduce suitable abstractions and handle increasing complexity, e.g., views on databases [1], to model applications [41], or to observe business processes [74]. In the following, different works covering views on workflows, i.e., process views, are discussed. Biton et al. [9] present how process views can be automatically generated based on user requirements. Thereby, they summarize multiple activities into single activities abstracting from the technical details. Furthermore, they introduce various properties to evaluate whether a process view is suitable or not. Reichert et al. [66] focus on the monitoring of workflows and how to properly visualize the process views using the *Proviado framework*. This includes defining abstract process view definitions by the user, which are then instrumented with collected data and visualized. Sonntag et al. [74] analyze existing concepts for process views on conventional workflows and extend them for scientific workflows. Scientific workflows lead to specific challenges due to their characteristics, e.g., incremental development. In addition to rewriting scientific workflows, Cohen-Boulakia et al. [16] also show how to use process views to analyze collected provenance data. Schumm et al. [70] utilize process views for ensuring the compliance of workflow executions. Therefore, they extract so-called *compliance fragments* and, e.g., show or hide certain details. To perform the transformation including or excluding information, a rule language is introduced. Finally,

also BPMN specifies sub-processes, which can be collapsed or expanded, providing a means to abstract technical details depending on the users [62]. However, this mechanism is not sophisticated enough for certain scenarios, as only the executed activities can be abstracted to a single sub-process. In contrast, it is not possible to visualize additional details, as required by our presented approach.

Conclusion and Future Work

Quantum workflows enable orchestrating the control and data flow between quantum and classical programs of hybrid quantum applications. Thereby, workflows provide various benefits, such as robustness, scalability, automated error handling, increased modularity, or simplified understanding of complex applications by providing a graphical notation. However, when executing the quantum and classical programs interleaved multiple times, the orchestration using workflows is inefficient due to the increased latency and queuing times. For such use cases, hybrid runtimes are provided, optimizing the execution by closely deploying the quantum and classical programs together and reducing the queuing times. In this paper, we presented a method to automatically detect workflow parts that can benefit from hybrid runtimes. Based on the workflow parts, equivalent hybrid programs are generated, and the workflow is rewritten to invoke them instead of orchestrating the workflow part. Additionally, provenance data about the execution of the hybrid programs are collected and can be used to instrument process views. These process views ease the monitoring and analysis of the workflow, as both the original and the rewritten workflow can be visualized, including the token flow and the current variable values. To validate the practical feasibility of our approach, we introduced a prototypical implementation and a case study comprising a quantum workflow from the humanities domain to which the analysis and rewrite method is applied. Finally, an evaluation was performed showing that the time required to analyze and rewrite a quantum workflow is small compared to the overall execution times. Further, for our experiments, the speed-up when using hybrid runtimes was about factor 2 compared to the execution of the original workflow directly accessing the quantum computers.

In future work, we plan to extend the detection and filtering steps by including non-functional requirements. These requirements are then also used to select a hybrid runtime when multiple are suitable for a workflow part. Additionally, we will incorporate new hybrid runtimes into our approach. Finally, we plan to investigate how to determine if other workflow parts comprising the interleaved execution of quantum and classical programs without a loop can benefit from hybrid runtimes and what factors must be considered.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was partially funded by the BMWK projects *EniQmA* (01MQ22007B) and *PlanQK* (01MK20005N).

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abiteboul S, Hull R, Vianu V. Foundations of databases. New York: Addison-Wesley; 1995. p. 8.
2. Agrawal R, Gunopulos D, Leymann F. Mining process models from workow logs. In: International conference on extending database technology. Berlin: Springer; 1998. p. 467–83.
3. Arute F, Arya K, Babbush R, Bacon D, et al. Quantum supremacy using a programmable superconducting processor. Nature. 2019;574(7779):505–10.
4. AWS: Amazon Braket Hybrid Jobs User Guide 2022. https://docs.aws.amazon.com/braket/latest/developerguide/braket-jobs.html.
5. Barzen J. From digital humanities to quantum humanities: potentials and applications. In: Quantum computing in the arts and humanities. Springer; 2021. ArXiv:2103.11825.
6. Barzen J, Leymann F, Falkenthal M, Vietz D, Weder B, Wild K. Relevance of near-term quantum computing in the cloud: a humanities perspective. Cloud Comput Serv Sci. 2021;1399:25–58.
7. Beisel M, Barzen J, Leymann F, Truger F, Weder B, Yussupov V. Patterns for quantum error handling. In: Proceedings of the 14th international conference on pervasive patterns and applications (PATTERNS), p. 22–30. Xpert Publishing Services (XPS) (2022).
8. Binz T, Breitenbücher U, Haupt F, Kopp O, Leymann F, Nowak A, Wagner S OpenTOSCA—a runtime for TOSCA-based cloud applications. In: Proceedings of the 11th international conference on service-oriented computing (ICSOC). Springer; 2013. p 692–695.
9. Biton O, Davidson SB, Khanna S, Roy S. Optimizing user views for workflows. In: Proceedings of the 12th international conference on database theory, pp. 310–323 (2009)
10. Breitenbücher U, Binz T, Képes K, Kopp O, Leymann F, Wettinger J. Combining declarative and imperative cloud application provisioning based on TOSCA. In: International conference on cloud engineering (IC2E). IEEE; 2014. p. 87–96.
11. Bucchiarone A, Marconi A, Pistore M, Raik H. Dynamic adaptation of fragment-based and context-aware business processes. In: Proceedings of the 19th international conference on web services (ICWS). IEEE, 2012. p. 33–41.
12. Camunda: Camunda BPMN Modeler 2022. https://camunda.com/products/camunda-bpm/modeler.

13. Camunda: Camunda BPMN Workflow Engine; 2022. https://camunda.com/products/camunda-bpm/bpmn-engine.
14. Cerezo M, Arrasmith A, Babbush R, Benjamin SC, Endo S, Fujii K, et al. Variational quantum algorithms. Nat Rev Phys. 2021;20:1–20.
15. CNCF: Kubernetes; 2022. https://kubernetes.io.
16. Cohen-Boulakia S, Biton O, Cohen S, Davidson S. Addressing the provenance challenge using ZOOM. Concurr Comput Pract Exp. 2008;20(5):497–506.
17. Cortese JA, Braje TM. Loading classical data into a quantum computer. arXiv:1807.02500 (2018).
18. De B. Api management. Berlin: Springer; 2017. p. 15–28.
19. Di Penta M, Esposito R, Villani ML, Codato R. Colombo, M., Di Nitto E. WS Binder: a framework to enable dynamic binding of composite web services. In: Proceedings of the 2006 international workshop on Service-oriented software engineering, 2006. p. 74–80.
20. Dumas M, La Rosa M, Mendling J, Reijers HA. Fundamentals of business process management, vol. 1. Berlin: Springer; 2013.
21. Eberle H, Unger T, Leymann F. Process fragments. In: On the move to meaningful internet systems (OTM). Berlin: Springer; 2009. p. 398–405.
22. Eder J, Liebhart W. Workflow recovery. In: Proceedings of the international conference on cooperative information systems. IEEE; 1996, p. 124–134.
23. Ellis CA. Workflow technology. Computer supported cooperative work. Trends Softw Ser. 1999;7:29–54.
24. Farhi E, Goldstone J, Gutmann S. A quantum approximate optimization algorithm. arXiv:1411.4028; 2014.
25. Farhi E, Harrow AW. Quantum supremacy through the quantum approximate optimization algorithm. arXiv:1602.07674; 2016.
26. Freire J, Koop D, Santos E, Silva CT. Provenance for computational tasks: a survey. Comput Sci Eng. 2008;10(3):11–21.
27. Gabor T, Sünkel L, Ritz F, Phan T, Belzner L, Roch C, Feld S, Linnhoff-Popien C. 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, 2020. p. 456–461.
28. HashiCorp: Terraform 2022. https://www.terraform.io.
29. Havlíček V, Córcoles AD, Temme K, Harrow AW, Kandala A, Chow JM, Gambetta JM. Supervised learning with quantum-enhanced feature spaces. Nature. 2019;567(7747):209–12.
30. Herschel M, Diestelkämper R, Ben Lahmar H. A survey on provenance: What for? What Form? What from? VLDB J. 2017;26(6):881–906.
31. IBM: IBM Quantum delivers 120x speedup of quantum workloads with Qiskit Runtime 2021. https://research.ibm.com/blog/120x-quantum-speedup.
32. IBM: IBM's roadmap for building an open quantum software ecosystem; 2021. https://research.ibm.com/blog/quantum-development-roadmap.
33. IBM: Qiskit runtime; 2022. https://github.com/Qiskit-Partners/qiskit-runtime.
34. IBM: Qiskit runtime documentation; 2022. https://qiskit.org/documentation/partners/qiskit_ibm_runtime/index.html.
35. IBM: Qiskit runtime limitations for custom programs; 2022. https://qiskit.org/documentation/partners/qiskit_ibm_runtime/tutorials/sample_vqe_program/qiskit_runtime_vqe_program.html.
36. Kandala A, Mezzacapo A, Temme K, Takita M, Brink M, Chow JM, Gambetta JM. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature. 2017;549(7671):242–6.
37. Karalekas PJ, Tezak NA, Peterson EC, Ryan CA, da Silva MP, Smith RS. A quantum-classical cloud platform optimized for variational hybrid algorithms. Quantum Sci Technol. 2020;5:2.
38. Képes K, Breitenbücher U, Sáez SG, Guth J, Leymann F, Wieland M. Situation-aware execution and dynamic adaptation of traditional workflow models. In: Proceedings of the 5th European conference on service-oriented and cloud computing (ESOCC). Springer; 2016. p. 69–83.
39. Khan SU, Awan AJ, Vall-Llosera G. K-means clustering on noisy intermediate scale quantum computers; 2019. arXiv:1909.12183.
40. Kopp O, Binz T, Breitenbücher U, Leymann F. Winery—a modeling tool for TOSCA-based cloud applications. In: Proceedings of the 11th international conference on service-oriented computing (ICSOC). Springer; 2013. p. 700–704.
41. Lara Jd, Guerra E, Sánchez-Cuadrado J. Abstracting modelling languages: a reutilization approach. In: Proceedings of the 24th international conference on advanced information systems engineering (CAiSE). Springer; 2012. p. 127–143.
42. LaRose R. Overview and comparison of gate level quantum software platforms. Quantum. 2019;3:25.
43. LaRose R, Coyle B. Robust data encodings for quantum classifiers. Phys Rev A. 2020;102(3): 032420.
44. Latva-Koivisto AM. Finding a complexity measure for business process models. Technical report, Helsinki University of Technology; 2001.
45. Leymann F. Supporting business transactions via partial backward recovery in workflow management systems. In: Datenbanksysteme in Büro, Technik und Wissenschaft. Springer; 1995, p. 51–70.
46. Leymann F, Barzen J. The bitter truth about gate-based quantum algorithms in the NISQ era. Quantum Sci Technol. 2020;5:4.
47. Leymann F, Barzen J. Hybrid quantum applications need two orchestrations in superposition: a software architecture perspective; 2021. arXiv:2103.04320.
48. Leymann F, Barzen J, Falkenthal , Vietz D, Weder B, Wild K. Quantum in the cloud: application potentials and research opportunities. In: Proceedings of the 10th international conference on cloud computing and services science. SciTePress; 2020. p. 9–24.
49. Leymann F, Roller D. Workflow-based applications. IBM Syst J. 1997;36(1):102–23.
50. Leymann F. Roller D. Method and computer system for generating process management computer programs from process models 2000. US Patent 6,011,917.
51. Leymann F, Roller D. Production workflow: concepts and techniques. New York: Prentice Hall PTR; 2000.
52. Liu J, Pacitti E, Valduriez P, Mattoso M. A survey of data-intensive scientific workflow management. J Grid Comput. 2015;13(4):457–93.
53. Lubinski T, Granade C, Anderson A, Geller A, Roetteler M, Petrenko A, Heim B. Advancing hybrid quantum-classical computation with real-time execution; 2022. arXiv:2206.12950.
54. Maciejewski FB, Zimborás Z, Oszmaniec M. Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography. Quantum. 2020;4:257.
55. McCaskey AJ, Dumitrescu EF, Liakh DI, Humble TS. Hybrid programming for near-term quantum computing systems. In: 2018 IEEE international conference on rebooting computing (ICRC). IEEE; 2018. p. 1–12.
56. McClean JR, Romero J, Babbush R, Aspuru-Guzik A. The theory of variational hybrid quantum-classical algorithms. New J Phys. 2016;18:2.
57. Michielsen K, Nocon M, Willsch D, Jin F, Lippert T, De Raedt H. Benchmarking gate-based quantum computers. Comput Phys Commun. 2017;220:44–55.

58. Müller R, Greiner U, Rahm E. AgentWork: a workflow system supporting rule-based workflow adaptation. Data Knowl Eng. 2004;51(2):223–56.
59. Mundbrod N, Grambow G, Kolb J, Reichert M. Context-aware process injection: enhancing process flexibility by late extension of process instances. In: On the move to meaningful internet systems (OTM). Springer; 2015. p. 127–145.
60. Nielsen MA, Chuang I. Quantum computation and quantum information. AAPT (2010).
61. OASIS: Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Organization for the advancement of structured information standards; 2007.
62. OMG: Business Process Model and Notation (BPMN) Version 2.0. Object Management Group; 2011.
63. Orban D. Templating and automatic code generation for performance with Python. Cahier GERAD G. 2011;2011:30.
64. Pelofske E, Bärtschi A, Eidenbenz S. Quantum volume in practice: what users can expect from NISQ devices. arXiv:2203.03816; 2022.
65. Preskill J. Quantum computing in the NISQ era and beyond. Quantum. 2018;2:25.
66. Reichert M, Bassil S, Bobrik R, Bauer T. The Proviado access control model for business process monitoring components. Enterprise Modell Inf Syst Arch. 2010;5(3):64–88.
67. Reiff-Marganiec S, Yu, HQ, Tilly M. Service selection based on non-functional properties. In: International conference on service-oriented computing (ICSOC). Springer; 2007, p. 128–138.
68. Rinderle-Ma S, Reichert M. Advanced migration strategies for adaptive process management systems. In: Proceedings of the 12th IEEE conference on commerce and enterprise computing. IEEE; 2010. p. 56–63.
69. Schumm D, Leymann F, Streule A. Process viewing patterns. In: Proceedings of the 14th international enterprise distributed object computing conference (EDOC). IEEE; 2010. p. 89–98.
70. Schumm D, Leymann F, Streule A. Process views to support compliance management in business processes. In: International conference on electronic commerce and web technologies. Springer; 2010. p. 131–142.
71. Shor PW. Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. IEEE; 1994. p. 124–134.
72. Sim S, Johnson PD, Aspuru-Guzik A. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. Adv Quantum Technol. 2019;2(12):1900070.
73. Song X, Dou W, Chen J. A workflow framework for intelligent service composition. Future Gener Comput Syst. 2011;27(5):627–36.
74. Sonntag M, Görlach K, Karastoyanova D, Leymann F, Malets P, Schumm D. Views on scientific workflows. In: Proceedings of the 10th international conference on perspectives in business informatics research (BIR), vol. 90. Springer; 2011. p. 321–335.
75. Steane A. Quantum computing. Re Progress Phys. 1998;61(2):117.
76. Tannu SS, Qureshi MK. 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; 2019. p. 987–999.
77. University of Stuttgart: Demo Video; 2022. https://www.youtube.com/watch?v=8nYsVCfuc7M.
78. University of Stuttgart: Evaluation Data; 2022. https://github.com/UST-QuAntiL/qprov-content/tree/main/workflow-analysis-and-rewrite.
79. University of Stuttgart: QuAntiL: Quantum Application Lifecycle Management; 2022. https://github.com/UST-QuAntiL.
80. University of Stuttgart: Quantum Workflow Use Cases; 2022. https://github.com/UST-QuAntiL/QuantME-UseCases/tree/master/2022-sncs.
81. Van der Aalst W. Process mining. Commun ACM. 2012;55(8):76–83.
82. Van Dongen BF, de Medeiros AKA. Verbeek H, Weijters, A., van der Aalst, W. The ProM framework: a new era in process mining tool support. In: International conference on applications and theory of Petri Nets. Springer; 2005. , p. 444–454.
83. Vietz, D., Barzen, J., Leymann, F., Weder, B., Yussupov, V.: 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). CEUR Workshop Proceedings, 2021. p. 1–12.
84. Wang M, Ramamohanarao K, Chen J. Trust-based robust scheduling and runtime adaptation of scientific workflow. Concurr Comput Pract Exp. 2009;21(16):1982–98.
85. Weder B, Barzen J, Beisel M, Leymann F. 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). SciTePress; 2022. p. 38–50.
86. Weder B, Barzen J, Leymann F. MODULO: modeling, transformation, and deployment of quantum workflows. In: Proceedings of the 25th international enterprise distributed object computing workshop (EDOCW). IEEE; 2021. p. 341–344.
87. Weder B, Barzen J, Leymann F, Salm M. Automated quantum hardware selection for quantum workflows. Electronics. 2021;10:8.
88. Weder B, Barzen J, Leymann F, Salm M, Wild K. QProv: a provenance system for quantum computing. IET Quantum Commun. 2021;2(4):171–81.
89. Weder B, Barzen J, Leymann F, Vietz D. Quantum software development lifecycle. Berlin: Springer; 2022. p. 61–83.
90. Weder B, Breitenbücher U, Képes K, Leymann F, Zimmermann M. Deployable self-contained workflow models. In: Proceedings of the 8th European conference on service-oriented and cloud computing (ESOCC). Springer; 2020. p. 85–96.
91. Weder B, Breitenbücher U, Leymann F, Wild K. Integrating quantum computing into workflow modeling and execution. In: Proceedings of the 13th IEEE/ACM international conference on utility and cloud computing (UCC). IEEE; 2020, p. 279–291.
92. Weigold M, Barzen J, Leymann F, Vietz D. Patterns for hybrid quantum algorithms. In: Proceedings of the 15th symposium and summer school on service-oriented computing (SummerSOC). Springer; 2021. p. 34–51.
93. Zimmermann M, Breitenbücher U, Képes K, Leymann F, Weder B. Data flow dependent component placement of data processing cloud applications. In: Proceedings of the IEEE international conference on cloud engineering (IC2E). IEEE; 2020. p. 83–94.