Analysis and Rewrite of Quantum Workflows: Improving the Execution of Hybrid Quantum Algorithms

Benjamin Weder[a], Johanna Barzen[b], Martin Beisel[c] and Frank Leymann[d]
Institute of Architecture of Application Systems, University of Stuttgart, Germany

Keywords: Quantum Computing, Hybrid Algorithms, Quantum Workflows, Workflow Rewrite, Hybrid Runtimes.

Abstract: With the rapid evolution of quantum computers and the emergence of different quantum cloud offerings, use cases from various application areas, such as chemistry or physics, can now be implemented and executed on real quantum computers. Thereby, the applications are typically hybrid, i.e., combine quantum and classical programs. Workflows enable the orchestration of these programs and provide advantages, such as robustness or reproducibility. However, different quantum algorithms require executing the quantum and classical programs in a loop with many iterations, leading to an inefficient orchestration through the workflow. For the efficient execution of such algorithms, hybrid runtimes are offered, combining the quantum and classical programs in a single hybrid program, optimizing the execution. However, this leads to a conceptional gap between the modeling benefits of workflow technologies, e.g., modularization, reuse, and understandability, and the efficiency improvements when using hybrid runtimes. To overcome this issue, we present a method to model all tasks explicitly in the workflow model and analyze the workflow to detect loops that can benefit from hybrid runtimes. Furthermore, corresponding hybrid programs are automatically generated, and the workflow is rewritten to use them. We validate the practical feasibility of our approach by a prototypical implementation.

1 INTRODUCTION

Quantum computers are made publicly available via the cloud by various providers, such as IBM or Rigetti (LaRose, 2019; Leymann et al., 2020). Thus, use cases from different application areas, e.g., computer science, physics, or chemistry, can be implemented and evaluated on real quantum computers (Weder et al., 2022). Thereby, quantum computing is expected to provide different benefits, such as speed-ups and higher accuracy for certain problems, e.g., for machine learning algorithms, or lower energy consumption compared to using classical infrastructures (Nielsen and Chuang, 2010; Barzen, 2021).

Today's quantum applications are typically hybrid, i.e., consist of quantum programs executed on a quantum computer and classical programs performing pre- and post-processing tasks using classical computers (McCaskey et al., 2018; Leymann and Barzen, 2020). Thereby, we focus on the gate-based quantum computing model, where quantum programs are represented by so-called *quantum circuits* (Nielsen and Chuang, 2010). The classical programs can, e.g., generate *state preparation circuits* based on the input data as a pre-processing step (LaRose and Coyle, 2020). These state preparation circuits are then prepended to the quantum circuit to be executed to initialize the register of the quantum computer with the required state (Leymann and Barzen, 2020). Another example of classical post-processing is mitigating errors occurring due to the noise of current quantum computers (Maciejewski et al., 2020; Weder et al., 2021c).

Thus, quantum applications require orchestrating a set of quantum and classical programs, which are often based on different programming languages, data formats, or invocation mechanisms (Leymann and Barzen, 2021; Weder et al., 2022). *Workflow technologies* are an orchestration approach that has been applied to integrate heterogeneous programs and services in various application areas, such as business process management or e-science (Leymann and Roller, 2000b; Liu et al., 2015). Thereby, workflows provide different benefits, e.g., scalability, robustness, or transactional processing (Leymann, 1995; Eder and Liebhart, 1996). Hence, workflows are also a promising means to orchestrate the programs comprising a hybrid quantum application (Weder et al., 2020b).

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

Weder, B., Barzen, J., Beisel, M. and Leymann, F.
Analysis and Rewrite of Quantum Workflows: Improving the Execution of Hybrid Quantum Algorithms.
DOI: 10.5220/0011035100003200
In *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER 2022)*, pages 38-50
ISBN: 978-989-758-570-8; ISSN: 2184-5042

However, various quantum algorithms require the interleaved execution of quantum and classical programs until a certain condition is met (McClean et al., 2016). With a high number of iterations, the orchestration by the workflow is getting inefficient, as it has to pass the control and data between the programs in each iteration (Leymann and Barzen, 2021). Efficient execution of these algorithms can be ensured by *hybrid runtimes* (Vietz et al., 2021) – an emerging type of runtime enabling the upload of quantum and classical programs together as so-called *hybrid programs*.

In this paper, (i) we introduce a method to enable the modeling of all tasks implementing hybrid quantum algorithms in the workflow model while increasing the efficiency through hybrid runtimes. Hence, instead of implementing a single hybrid program comprising all tasks and invoking it by the workflow, the modularity is retained, increasing the reuse and understandability. However, to benefit from the advantages of hybrid runtimes, (ii) we automatically analyze the workflow model to find suitable optimization candidates, generate corresponding hybrid programs, and rewrite the workflow to invoke them. To prove the practical feasibility of our method, (iii) we present a prototypical implementation based on the *MODULO framework* (Weder et al., 2021a) and *Qiskit Runtime*, a hybrid runtime provided by IBM (IBM, 2021c). Finally, (iv) we showcase the analysis and rewrite method based on an exemplary quantum workflow.

The remainder of the paper is structured as follows: In Section 2, fundamentals and the problem statement are described. Then, Section 3 presents the method to model quantum workflows independent of a certain runtime, as well as their automated analysis and rewrite to improve the execution of hybrid algorithms. In Section 4, the prototype implementing the presented method is introduced. Subsequently, Section 5 validates the method with a case study, and Section 6 evaluates its performance. Section 7 discusses possible extensions, as well as consequences regarding the monitoring and analysis of quantum workflows. Finally, the related work is discussed in Section 8, and Section 9 concludes the paper.

2 FUNDAMENTALS & PROBLEM STATEMENT

In the following, we introduce fundamentals about hybrid quantum algorithms and hybrid runtimes. Furthermore, the need for workflow technology to orchestrate the different software artifacts implementing a quantum application is motivated. Finally, we present the problem statement of this work.

2.1 Hybrid Algorithms & Runtimes

Today‘s quantum computers provide a limited number of qubits and are affected by noise from different sources, restricting the number of operations that can be successfully executed (Leymann and Barzen, 2020). Therefore, they are often referred to as *Noisy Intermediate-Scale Quantum (NISQ)* devices (Preskill, 2018). Due to these limitations, different quantum algorithms, such as the *HHL algorithm* (Harrow et al., 2009) for solving linear systems of equations, can currently not be executed on practically useful problems (Leymann and Barzen, 2020).

To overcome this issue and to already benefit from quantum computing during the NISQ era, so-called *hybrid algorithms* are used (Leymann and Barzen, 2021; Weder et al., 2022). Hybrid algorithms split the computational tasks and distribute them over quantum and classical computers (Weigold et al., 2021). The goal of this splitting is to derive shallow quantum programs using only a small number of qubits and operations (Cerezo et al., 2021). Thus, they can be successfully executed on today‘s quantum computers, with their short decoherence times and high error rates (Salm et al., 2020; Preskill, 2018). Different hybrid algorithms also require executing the quantum and classical programs in a loop (McClean et al., 2016). Examples relying on this approach are the *quantum approximate optimization algorithm (QAOA)* (Farhi et al., 2014), which can be used for solving various optimization problems, or the *variational quantum eigensolver (VQE)* (Kandala et al., 2017) to determine eigenvalues of a matrix.

Queue-based access to quantum computers is unsuitable when executing hybrid algorithms comprising a loop of quantum and classical processing (Leymann and Barzen, 2021; Karalekas et al., 2020). This results in submitting a request to the queue in each iteration, increasing the latency of the hybrid algorithm significantly. Instead, access to the quantum computer has to be reserved for the whole duration of the loop, which different quantum hardware providers enable by booking a corresponding time slice (LaRose, 2019). However, the data must still be transmitted between the quantum and the classical programs, e.g., executed on a user device or some cloud offering, in each iteration, increasing the latency (IBM, 2021a). To avoid this, new offerings are emerging, such as the *Qiskit Runtime* (IBM, 2021c) or *Amazon Braket Hybrid Jobs* (AWS, 2021), to which we refer to as hybrid runtimes. Hybrid runtimes allow uploading the quantum and classical programs of a hybrid algorithm together for execution. Then, the classical programs are provisioned close to the used quantum computer, and the communication between them is optimized.

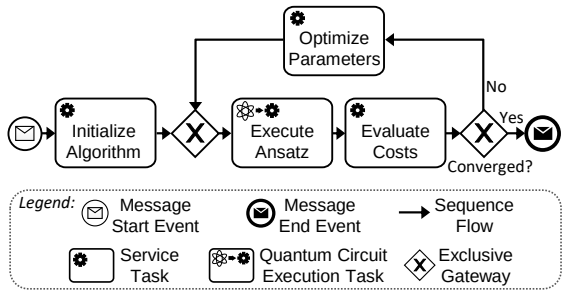

Figure 1: Quantum workflow executing a variational quantum algorithm (adapted from (Vietz et al., 2021)).

2.2 Quantum Workflows & QuantME

The different programs implementing a hybrid algorithm have to be orchestrated and required data must be passed between them (Weder et al., 2021a). Further, quantum applications may comprise multiple hybrid algorithms and additional classical programs independent of a quantum algorithm, e.g., loading data from a database or interacting with the user (Weder et al., 2022). An example can be found in Leymann and Barzen (2021). By utilizing workflow technologies to orchestrate the programs, quantum applications can benefit from their advantages, such as robustness, scalability, or the possibility to interrupt the execution (Ellis, 1999; Leymann and Roller, 2000b). Thereby, the collection of required activities, their partial order, and the data flow between them is specified in a *workflow model*. This workflow model can be uploaded to a workflow engine for execution.

However, the modeling of quantum workflows is complex and requires quantum-specific expertise, as well as knowledge about workflow technologies (Vietz et al., 2021; Weder et al., 2022). Furthermore, existing workflow languages, such as *BPMN* (OMG, 2011) or *BPEL* (OASIS, 2007), do not incorporate explicit modeling constructs for the different frequently occurring pre-processing, quantum program execution, and post-processing tasks with their specific characteristics (Weder et al., 2021a). To ease the modeling of these tasks and to increase the reuse of their implementations, we introduced the *quantum modeling extension (QuantME)* (Weder et al., 2020b), which can be applied to various imperative workflow languages. QuantME incorporates new modeling constructs for different tasks, e.g., the *data preparation task*, generating a state preparation circuit depending on the input data (LaRose and Coyle, 2020), or the *quantum circuit execution task*, executing a quantum circuit. All QuantME modeling constructs define a set of configuration properties, which can be used to customize them, e.g., by specifying which encoding to use to generate the state preparation circuit.

In Figure 1, an exemplary quantum workflow implementing the general structure of a so-called *variational quantum algorithm (VQA)*, such as VQE, is shown (Cerezo et al., 2021). VQAs are special kinds of hybrid algorithms relying on a parameterized quantum circuit, called *ansatz*, for which the parameters are optimized in each iteration of quantum and classical processing (Sim et al., 2019; Weigold et al., 2021). The quantum workflow starts when a message with the input data is received. Then, it initializes the algorithm by loading the ansatz and calculating the initial parameters. Afterwards, the hybrid loop is entered, executing the ansatz on a quantum computer and evaluating the cost function encoding the solutions of the problem to solve with the retrieved results. If the costs converge to an optimum or another termination condition is met, the workflow returns the result to the user. Otherwise, the next iteration is entered by optimizing the parameters based on the previous results.

2.3 Problem Statement

As described in the previous section, quantum applications can be modeled using workflows to benefit from their advantages. In addition to the advantages already discussed, their graphical notation also eases the understanding of implemented hybrid algorithms, even for non-quantum experts. Further, they increase the modularization by splitting the functionality into independent services. Hence, the implementations of the tasks can easily be exchanged, e.g., to test another optimizer in the example from Figure 1. However, the orchestration of loops consisting of quantum and classical processing by the workflow is inefficient. Instead, the workflow should be automatically analyzed and rewritten to use hybrid runtimes for the execution of such hybrid loops. Thus, our main research question (RQ) for this work can be formulated as follows:

> **RQ:** *"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 the RQ by splitting it into the following questions tackling the two main challenges:

> **Sub-RQ 1:** *"How can workflows be analyzed to detect hybrid loops, which can be executed more efficiently using hybrid runtimes?"*

> **Sub-RQ 2:** *"How can programs for hybrid runtimes automatically be generated based on such loops, and workflows be rewritten to use them?"*

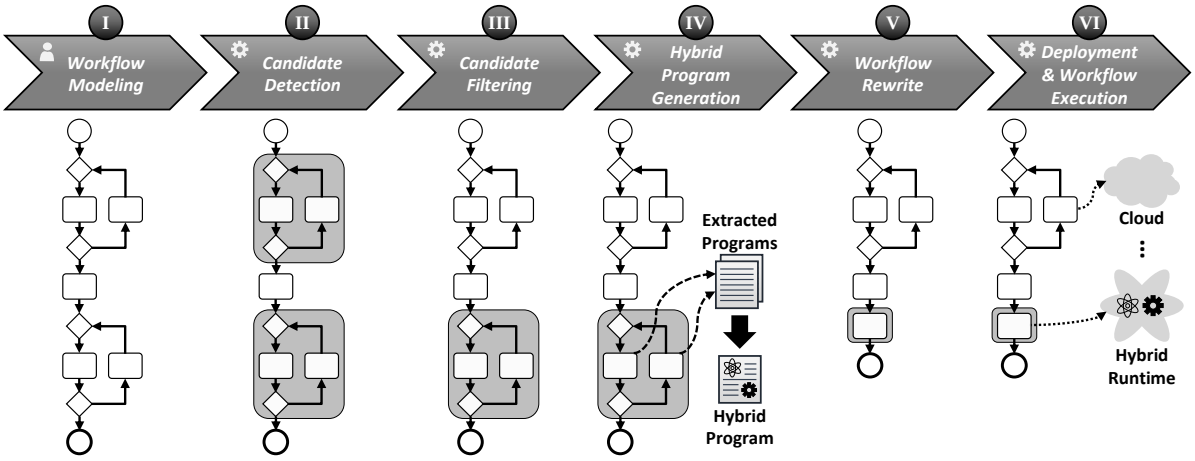

Figure 2: Overview of the method to analyze and rewrite quantum workflows to benefit from hybrid runtimes.

3 QUANTUM WORKFLOW ANALYSIS AND REWRITE

In this section, we present our method to analyze and rewrite quantum workflows to use hybrid runtimes for the execution of hybrid loops. The method consists of six steps, which are discussed in the following subsections. Figure 2 gives an overview of the method and the included steps. Thereby, steps two and three address Sub-RQ 1 and steps four and five Sub-RQ 2.

3.1 Workflow Modeling

In the first step, the quantum workflow is modeled, comprising all required quantum and classical tasks, their partial order, and the data flow between them (Leymann and Roller, 2000b; Dumas et al., 2013). In addition to the native modeling constructs of the utilized workflow language, QuantME modeling constructs are used to define the quantum-specific tasks. Although QuantME can be applied to different imperative workflow languages (see Section 2.2), we use the BPMN (OMG, 2011) concepts for all further considerations, as it is widely used and provides a well-known graphical notation. The result of the workflow modeling step is a workflow model orchestrating the tasks of one or multiple hybrid algorithms, as well as additional classical tasks if needed.

3.2 Candidate Detection

Next, the quantum workflow is automatically analyzed to detect workflow fragments orchestrating the interleaved execution of quantum and classical processing, which are candidates for an optimized execution using a hybrid runtime. These workflow fragments must satisfy three conditions: (i) They must contain one or more quantum circuit execution tasks, the QuantME modeling construct to execute quantum circuits on a quantum computer (Weder et al., 2020b). Otherwise, no quantum computers are needed, and thus, also a hybrid runtime is not useful. (ii) Furthermore, the workflow fragment must comprise classical processing, e.g., modeled using a script or service task in BPMN. If the workflow fragment only contains quantum processing and no classical programs, the quantum circuits can also be executed using batch processing, and no hybrid runtime is needed. Another optimization for multiple quantum circuit execution tasks can be to concatenate the different quantum circuits into one circuit and execute only this circuit. However, the quantum circuit concatenation leads to an increased depth of the resulting circuit, which can be unsuitable for today's quantum computers (Salm et al., 2020). Thus, a thorough analysis of the circuit, as well as the available quantum computers is required, which we plan to incorporate into the analysis and rewrite method as part of our future work. (iii) Finally, the quantum and classical processing must be executed interleaved multiple times to benefit from hybrid runtimes. Without the interleaved execution, access to a quantum computer using a queue or reserving a time slice is usually sufficient (Vietz et al., 2021). This can either be modeled by several quantum and classical tasks or a loop in the workflow model. As the efficiency gain of hybrid runtimes usually increases with the number of iterations, we restrict the candidates to loops and leave other scenarios as future work. Such loops are typically represented by two corresponding splitting and merging gateways in BPMN, but they can also

be modeled, e.g., as a single gateway or by directly connecting activities using conditional sequence flow.

3.3 Candidate Filtering

In this step, the candidates are filtered based on their properties and the characteristics of the hybrid runtimes, as well as their supported hybrid programs. This means the filtering is applied for each available hybrid runtime independently, and different restrictions apply. Thus, a corresponding system supporting the analysis and rewrite method has to be plugin-based and extensible for new hybrid runtime offerings (see Section 4.1). However, the restrictions can be grouped into the following categories, which must be analyzed when adding a new hybrid runtime:

(1) **Programming Language:** The set of supported programming languages for the hybrid programs is typically restricted. This also limits the allowed languages of the quantum and classical programs implementing the tasks within the candidates, as they must be merged into the hybrid program. For example, only Python may be allowed for classical and OpenQASM for quantum programs.

(2) **Activity and Task Types:** Not all activity and task types allowed by the used workflow language can be represented in a hybrid program. For example, human tasks are not suitable, as they often result in long delays contradicting the idea of hybrid runtimes to have fast iterations between classical and quantum processing. Furthermore, transactions can not be properly handled in hybrid runtimes and are not admitted in optimization candidates. For sub-processes, the candidate filtering has to be applied recursively for all contained modeling constructs. Finally, modeling constructs such as the BPMN call activity invoking other globally defined tasks or workflows can also be handled recursively if their required definitions are accessible for the analysis and rewrite method.

(3) **Events:** Similar to task types, not each supported event can be mapped to a hybrid program. While some of the events, such as catching message or signal events, also end up in impractical delays, other events result in technical difficulties when generating the hybrid programs. For example, a throwing compensation event can be mapped to an API call from the hybrid program to the workflow engine executing the workflow to trigger compensation. However, this must then be supported by the hybrid runtime and the workflow engine.

(4) **Gateways:** Some of the gateways used in optimization candidates might not be supported by all available hybrid runtimes. For example, because they are event-based or have global semantics, such as the inclusive gateway in BPMN.

(5) **Quantum Computer Provider:** Finally, the quantum circuit execution task within the candidate allows specifying a certain provider or even a specific quantum computer to use for the execution of the quantum circuit. Thus, only a hybrid runtime supporting this provider or quantum computer can be used then to execute the hybrid loop.

In addition to this runtime-specific filtering, a general requirement for the rewrite is the availability of the source code for the quantum and classical programs implementing the tasks within the candidates. Otherwise, the generation of a corresponding hybrid program is not possible. This can, e.g., be achieved for script tasks by packaging the scripts with the workflow model or for service tasks by attaching deployment models with the required code to them (Weder et al., 2020a). However, it is usually not possible for services from external providers, e.g., available through a service registry (De, 2017).

Finally, if multiple hybrid runtimes are suitable for a candidate, the user can be asked for a selection, or it is automatically selected based on non-functional requirements, such as costs or trust in the provider.

3.4 Hybrid Program Generation

After determining a suitable hybrid runtime for a candidate, the hybrid program implementing the functionality of the workflow fragment is generated (Leymann and Roller, 2000a). For this, the programs corresponding to the tasks of the workflow fragment are extracted. Further, code snippets implementing the functionality of the contained gateways and events are generated. This can be achieved by providing templates for the supported gateways and events and then instrumenting them with the variable parts, e.g., the condition for a gateway (Orban, 2011). Next, the input and output parameters for the hybrid program have to be determined by analyzing the workflow. Thereby, the input parameters of all modeling constructs are collected and then filtered if they are the output of previous modeling constructs within the workflow fragment. Similarly, the output parameters are detected by merging the output parameters of the constructs within the workflow fragment and filtering them if they are not used as input for another construct outside the workflow fragment. Based on the determined input and output parameters, the interface of the hybrid program is created. Finally, the extracted programs and generated code snippets are merged into the hybrid program depending on the sequence flow

of the workflow fragment. Additionally, variables are introduced and passed within the hybrid program to reflect the data flow defined in the workflow fragment.

3.5 Workflow Rewrite

In the fifth step, the workflow is adapted to invoke the hybrid programs generated in the previous step. For this, a new service task is inserted into the workflow model to kick off the execution within the hybrid runtime through the corresponding API. Further, the in- and outgoing sequence and data flow of the replaced workflow fragment are redirected to the inserted service task. In the same way, boundary events of the tasks within the workflow fragment can be added to the service task if, e.g., the hybrid runtime supports terminating the execution when an interrupting boundary event is triggered. Otherwise, the rewrite must be aborted to avoid changing the semantics.

In addition, deployment models are automatically generated for each hybrid program. These deployment models are then attached to the service tasks to enable a self-contained packaging of the workflow model and all required data and code (Weder et al., 2020a). In contrast to directly deploying the hybrid programs, this can result in lower costs when deploying them on-demand before the workflow execution.

3.6 Deployment & Workflow Execution

In the last step, the required services for the workflow execution that are not always running are deployed. This includes services that are hosted on classical infrastructure, e.g., in the cloud, using an HPC, or a local workstation (Weder et al., 2021a). Further, the deployment of the generated hybrid programs to the hybrid runtimes is performed. For hybrid programs, the created deployment models from the previous step are utilized. All other services can also be deployed automatically if the required deployment models are attached to the activities (Weder et al., 2020a). Depending on the kind of deployment models, different deployment systems can be used, such as *Kubernetes* (CNCF, 2021), *Terraform* (HashiCorp, 2021), or the *OpenTOSCA Container* (Binz et al., 2013). After deploying the services, they are bound to the workflow by updating the endpoint information in the workflow model with the details about the deployed services. Finally, the updated workflow model is uploaded to a workflow engine and can be instantiated.

4 PROTOTYPICAL VALIDATION

In this section, we present the system architecture supporting the analysis and rewrite of quantum workflows and discuss our prototypical implementation.

4.1 System Architecture

Figure 3 presents the overall system architecture of our framework. Thereby, the *MODULO framework* (Weder et al., 2021a) to model, transform, and deploy quantum workflows is extended by additional components and services to support the analysis and rewrite method introduced in Section 3. Existing unchanged components are light, expanded components are grey, and newly added components are dark.

The *QuantME Transformation Framework* is a graphical BPMN modeler supporting QuantME, which is based on the Camunda modeler (Camunda, 2021a). It comprises the plugin-based *Workflow Analyzer*, which enables candidate detection and filtering. Thereby, new plugins can be added for a hybrid runtime defining the corresponding restrictions for the filtering (see Section 3.3). The *QuantME Modeler* component enables the graphical modeling of the QuantME and the native BPMN modeling constructs. It was extended to trigger the workflow analysis and rewrite, as well as visualization of found candidates to enable a user selection if multiple hybrid runtimes are suitable. All deployment-related functionality is implemented in the *Deployment Orchestrator*. For this, it uses *Winery* (Kopp et al., 2013), a graphical modeling tool based on the *TOSCA standard* (OASIS, 2013), and the *OpenTOSCA Container* (Binz et al., 2013), a TOSCA-compliant runtime, to deploy the required services. Furthermore, it automatically binds the deployed services with the workflow and uploads the workflow model to the *Camunda BPMN Engine* (Camunda, 2021b). The Deployment Orchestrator was extended to create the deployment models for the generated hybrid programs by instrumenting the connected Winery and attaching them to the new service tasks of the rewritten workflow model.

Rewriting quantum workflows after their analysis is done by the new *Workflow Rewriter* component. Thus, it extracts the quantum and classical programs for each candidate and sends them to the *Hybrid Runtime Handlers* (depicted on the left) for each hybrid runtime that was not filtered by the Workflow Analyzer. The Hybrid Runtime Handlers are external components that generate a hybrid program for a certain runtime based on the given quantum and classical programs. This is exemplarily shown by the *Qiskit Runtime Handler* generating hybrid programs

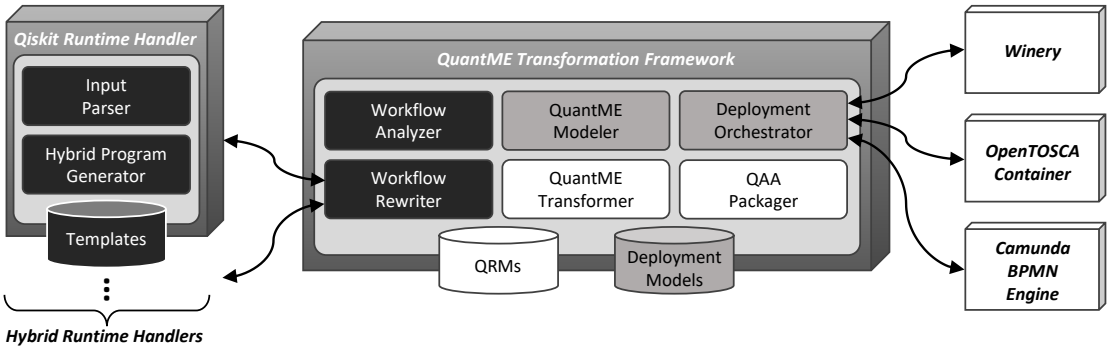

Figure 3: System architecture of the extended MODULO framework.

for Qiskit Runtime (IBM, 2021c) by IBM. The quantum and classical programs are forwarded to the *Input Parser*, which parses the programs to extract the code that must be merged into the hybrid program. Then, the hybrid program is created by the *Hybrid Program Generator*, using the extracted code, as well as the sequence and data flow passed by the Workflow Rewriter. Further, required code snippets implementing the functionality of events and gateways within the workflow fragment are generated based on a *Templates* repository (see Section 3.4). The created hybrid program is sent back to the Workflow Rewriter, adapting the workflow model to trigger its invocation.

Finally, the QuantME Transformation Framework consists of two unchanged components: First, the *QuantME Transformer* enables the transformation of workflow models comprising QuantME modeling constructs to native workflow models (Weder et al., 2020b). This retains the portability of the workflow models and avoids the need to extend existing workflow engines to understand the QuantME modeling constructs. For the transformation, it uses reusable workflow fragments implementing the required functionalities, which are stored as so-called *QuantME Replacement Models (QRMs)* in a corresponding repository. Second, the *QAA Packager* exports workflows with all needed information, e.g., deployment models, code, or data, in a self-contained *quantum application archive (QAA)*. Therefore, only this self-contained archive has to be transferred into the target environment for the workflow execution.

4.2 Prototypical Implementation

The QuantME Transformation Framework is implemented as a desktop application using the *Electron framework*, consisting of a graphical user interface and a Node.js backend. To prove the practical feasibility of our method, we implemented the Qiskit Runtime Handler as an exemplary Hybrid Runtime Handler. However, with the emergence of hybrid runtimes from different providers, we plan to extend the framework correspondingly. Thereby, the Qiskit Runtime Handler is realized in Python. Our prototypical implementation is publicly available as an open-source project on Github (University of Stuttgart, 2021c).

For the execution of generated hybrid programs, the presented prototype is based on Qiskit Runtime (IBM, 2021c), which is accessible over the quantum cloud offering *IBMQ*. Qiskit Runtime is currently in beta mode. Thus, some restrictions apply to its usage, as well as the supported hybrid programs, which are planned to be resolved in future releases (IBM, 2021d). However, these restrictions also influence the functionality of our prototype. For example, the Qiskit Runtime programs are based on Python, and the execution environment within the runtime only provides a limited set of available dependencies. This means, custom dependencies, e.g., including other optimizers, can currently not be installed. Hence, also the quantum and classical programs within the candidates are not allowed to use such dependencies, and otherwise, the rewrite must be aborted. Another example is that the hybrid programs must only consist of one file. Therefore, all code has to be merged into this file, reducing the understandability. However, we plan to resolve these issues as part of future work once Qiskit Runtime is released with full functionality.

5 CASE STUDY

To showcase the practical feasibility of the analysis and rewrite method, we present an exemplary quantum workflow from the domain of the quantum humanities (Barzen, 2021). Thereby, clustering and classification is performed on costume data. The workflow model is shown at the top of Figure 4.

The workflow is instantiated when a request message with the required input is received. For the sake of simplicity, this input contains an URL to

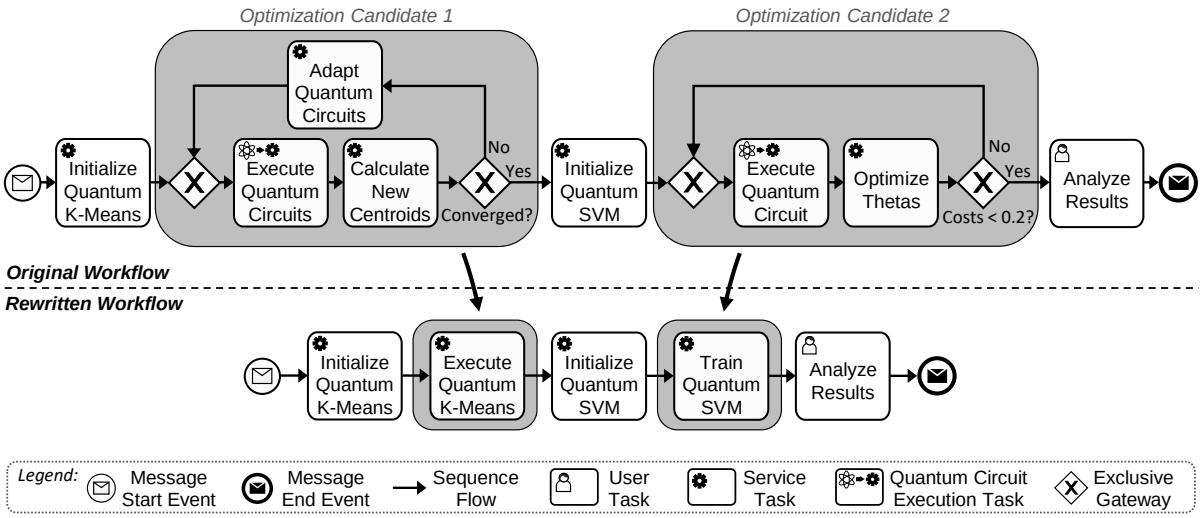

Figure 4: An exemplary quantum workflow comprising two hybrid loops which can be optimized using a hybrid runtime.

the already pre-processed costume data, i.e., categorical data is transformed to numerical data, and the dimension is reduced for the machine learning algorithms (Barzen, 2021). However, these steps can also be included, e.g., as service tasks in the workflow. Next, the clustering starts, whereby the *quantum k-means algorithm* (Khan et al., 2019) is orchestrated by multiple tasks. In the first task, the quantum k-means algorithm is initialized, i.e., the data is loaded from the URL, random initial centroids are chosen, and the quantum circuits for the algorithm are generated. Afterwards, the algorithm enters a hybrid loop, executing the generated quantum circuits on a quantum computer, and calculating the new centroids based on the measurement results in a classical service task. If the difference between the old and new centroids is larger than a given threshold, the quantum circuits are adapted to the new centroids, and the loop is executed again. Once the clustering converges, a classifier is trained based on the results using a *variational quantum support vector machine* (Havlíček et al., 2019). Thereby, an ansatz (see Section 2.2) and an initial parameterization are generated. Then, the quantum circuit is executed and the parameters theta are optimized based on the results. Furthermore, the cost function is evaluated with the new parameters and if the costs are higher than the threshold 0.2, the loop is entered again. Finally, if the costs are below the threshold, the result can be analyzed in a user task and is returned to the user by the message end event.

When analyzing the workflow using the QuantME Transformation Framework, it detects two hybrid loops as optimization candidates, which are visualized as gray boxes in Figure 4. Thereby, the initialization tasks for the two algorithms are not part of the hybrid loop and are, therefore, also not contained in the candidates. For both candidates, it is checked if they can be optimized using Qiskit Runtime. As the candidates do not contain, e.g., invalid events (see Section 3.4) and the source code is available as part of deployment models attached to the activities, they are not filtered, and the rewriting can be performed. The resulting rewritten workflow is shown at the bottom of Figure 4. Thereby, it comprises five tasks, which are executed in sequence. The two service tasks to initialize the hybrid algorithms, as well as the user task to analyze the results, are unchanged. However, the two remaining service tasks (surrounded by gray boxes) both replace one of the optimization candidates of the original workflow model and invoke the generated hybrid programs instead when they are executed.

The discussed use-case together with a detailed description of how to set up and use the framework can be found on Github (University of Stuttgart, 2021d). Further, a short demonstration video is available on YouTube (University of Stuttgart, 2021a).

6 EVALUATION

In this section, we present the results of our evaluation regarding the time required to detect candidates within different workflow models, as well as the generation of the corresponding hybrid programs and the workflow rewrite. Further, we compare the workflow runtime using the original workflow of our case study and the rewritten workflow after applying our method.

Table 1: Runtime Evaluation of the Analysis and Rewrite Method.

Workflow ID	# Activities	# Candidates	Avg. Activities per Candidate	Candidate Detection	Program Generation & Rewrite
1	8	2	2.5	0.30 s	108.00 s
2	50	2	2.5	0.44 s	108.60 s
3	50	4	2.5	0.48 s	219.67 s
4	50	6	2.5	0.75 s	323.45 s
5	50	2	5	0.40 s	314.54 s
6	50	4	5	0.59 s	631.51 s
7	50	6	5	0.73 s	929.19 s

6.1 Runtime of the Analysis and Rewrite Method

To evaluate the performance of our analysis and rewrite method, we modeled seven differently sized workflow models. Table 1 shows the required time to detect the candidates, as well as the time to generate the hybrid programs and rewrite the workflows for the different scenarios. Thereby, the workflow models are characterized by three attributes: (i) the number of activities within the workflow model, (ii) the number of candidates that can be detected, and (iii) the average activities per candidate. The number of activities increases the search space for the candidate detection and can influence the required time. Furthermore, the number of candidates corresponds to the number of hybrid programs that have to be generated for a workflow model. The program generation time also depends on the number of activities per candidate, as these programs have to be analyzed and merged into the hybrid program. Finally, the language and length of the different programs, i.e., their lines of code, have an impact on this step. Thus, we use Python programs with a similar length of around 200 lines of code to implement the activities for all workflow models. Thereby, the target hybrid runtime for the program generation is Qiskit Runtime.

The first workflow model in Table 1 is the case study presented in Section 5. As shown in Figure 4, the workflow model contains 8 activities and 2 candidates. Further, one of the candidates comprises 3 and the other 2 activities, leading to an average number of activities per candidate of 2.5. The workflow with ID 2 extends this workflow with additional activities, which are not related to the execution of hybrid algorithms. Thereby, the candidate detection for workflow 2 requires slightly more time than for workflow 1 because of the increased search space. In contrast, the time for the program generation and workflow rewrite is almost equal due to the same number and size of the candidates. As the time for the analysis is negligible compared to the program generation and rewriting, all further workflow models add additional candidates or adapt their size but do not change the overall number of activities. Thus, workflow 3 and 4 add two candidates each but retain their average size. It can be seen that the time for the program generation and rewrite grows linearly with the number of candidates. The reason for this is the sequential processing of the candidates in the current prototype. However, this can be parallelized in the future to improve the performance when rewriting workflows with a large number of candidates. Finally, for workflow 5 to 7, the average number of activities per candidate is increased from 2.5 to 5. Similar to workflows 2 to 4, the time increases linearly with the number of candidates. However, when doubling the average activities per candidate, the required time grows approximately by a factor of 2.9. The reason for this is that in addition to the doubled lines of code to analyze and merge, the determination of the interface of the resulting hybrid program and the sequence and data flow between the different program parts gets more complicated.

In general, the required initial time to analyze and rewrite a quantum workflow is rather short compared to the assumed runtime of large workflows, which often takes multiple hours, days, or even years (Leymann and Roller, 2000b). Furthermore, the analysis and rewrite must only be performed once, and then, the workflow can be executed multiple times.

The presented measurements were performed on a computer running Windows 11 64-bits with an Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz processor and 40 GB of RAM. Thereby, the median based on 50 measurements for each workflow is calculated. The collected raw data of each measurement is available on Github (University of Stuttgart, 2021b).

6.2 Workflow Runtime Comparison

In the following, we compare the time to execute the workflow presented as case study in Section 5 before and after applying our method. For this, Qiskit Runtime was used as the hybrid runtime to execute

the hybrid programs of the rewritten workflow. Furthermore, *ibmq_lima* was the target quantum computer for both the hybrid programs and the quantum programs of the original workflow. Thereby, we executed both workflows ten times, leading to a median execution time for the original workflow of 9617 s and the rewritten workflow of 2634 s. The queue of the quantum computer was empty during our execution. Therefore, the quantum programs of the original workflow were not required to wait in the queue until other jobs were completed. This means the accomplished speed-up is solely achieved by the optimized execution and reduced latency of Qiskit Runtime. Other jobs in the queue can further increase the speed-up. In both cases, the workflow engine was executed on the computer described in the previous subsection. Finally, also the required classical programs were executed on this computer. As part of our future work, we plan to further evaluate the achieved speed-up by executing various workflow models and utilizing different quantum computers and hybrid runtimes.

7 DISCUSSION

In the following, we discuss how the detection of suitable workflow fragments can be improved and how different kinds of runtimes can be incorporated into the presented method. Further, the problem of monitoring and analyzing rewritten workflows, as well as automated quantum computer selection, are explored.

In addition to the hybrid runtimes discussed in Section 2.1, a set of different pre-build runtimes is expected to evolve, as, e.g., announced by IBM (IBM, 2021b). These runtimes could, for example, incorporate a workflow engine or an HPC in combination with quantum computers. Hence, they can be integrated into our method by adding corresponding detection and filtering rules. Further, the required software artifacts, such as sub-workflows executed within the runtime, must then be generated automatically.

The candidate detection and filtering are currently only based on the structure of the workflows and the characteristics of the hybrid runtimes. However, this can be improved by attaching policies and non-functional requirements to the workflow activities (Di Penta et al., 2006). They then have to be taken into account for the candidate detection and filtering, as well as the selection of a concrete runtime to use. Such policies could, e.g., define that the execution time for a certain task or sub-process is especially important or that it comprises confidential data that is not allowed to be transferred to specific providers. Another example would be to annotate that a certain algorithm uses so-called *warm starting* (Egger et al., 2021), which requires elaborated classical pre-processing, indicating that a runtime combining an HPC and a quantum computer is preferable.

Due to the inherently probabilistic nature of quantum computing, it is of vital importance to collect as detailed information as possible about quantum applications and their execution (Weder et al., 2021c; Leymann and Barzen, 2020). This information helps to find the origins of errors, as well as to increase reproducibility and understandability. The systematic and automated collection and analysis of such information are referred to as *provenance* (Herschel et al., 2017). In the workflow domain, it is used to enable the monitoring of running workflow instances and to store it in the long-term in the so-called *audit trail* (Waters et al., 2004). This audit trail can then be analyzed to, e.g., improve the workflow models. However, the rewrite of the workflow complicates the monitoring and analysis, as the workflow model then only contains one task hiding all details instead of the whole hybrid loop. Hence, to overcome this issue, we plan to provide *provenance views* in the workflow engine as part of future work, enabling to switch between the rewritten and original workflow model for monitoring and analysis. This comes with additional challenges, such as extracting intermediate data during the hybrid program execution to support the live monitoring.

By defining a provider or even a specific quantum computer to use in quantum circuit execution tasks, the rewrite possibilities are restricted, as only corresponding hybrid runtimes can be used then (see Section 3.3). Thus, this should be avoided if not needed due to non-functional requirements, e.g., the trust in the provider. Then, the rewrite can be performed for an arbitrary suitable hybrid runtime, and the concrete quantum computer to use can be selected from the set of quantum computers available through the hybrid runtime. For this selection, tools, such as the *NISQ Analyzer* (Salm et al., 2020), can be used, analyzing the quantum circuit and quantum computer properties and selecting a suitable one for the execution.

8 RELATED WORK

Different research works propose methods and tools to analyze and rewrite workflow models during design time, as well as dynamically adapting them during runtime, which are discussed in this section.

Bucchiarone et al. (2012) introduce an approach to model abstract tasks with annotated goals, preconditions, and effects and dynamically replace them with fine-grained workflow fragments. Mundbrod et al.

(2015) present a method to rewrite workflows by injecting workflow fragments depending on the context, e.g., the available resources for a computation. Képes et al. (2016) describe a similar approach to dynamically select a workflow fragment for a required operation depending on the current situation. In contrast, we transform from the fine-grained workflow model to a more abstract one. Furthermore, we do not use available workflow fragments with the corresponding implementations but generate the required programs automatically based on the rewritten workflow. Cohen-Boulakia et al. (2012) present a method to rewrite scientific workflows to satisfy so-called series-parallel structures enabling efficient processing of provenance data. They also do not generate new programs but only rearrange vertices in the graph representing the scientific workflow. Furthermore, different approaches create new workflow models or adapt existing ones based on the audit trail using process mining (Agrawal et al., 1998; Van Der Aalst, 2012).

AristaFlow (Rinderle-Ma and Reichert, 2010) is a so-called adaptive workflow engine supporting the adaptation of running workflow instances. Furthermore, it enables their migration to new versions of the instantiated workflow model. Another example of such an adaptive workflow engine is *AgentWork* (Müller et al., 2004). Instead of modeling all required alternative sequence flows, it automatically adapts the workflows in exception cases by adding or removing activities. However, for both workflow engines, extensions are required to support the presented workflow analysis and rewrite method. Thus, this reduces the portability of the workflows, which is avoided in our approach by rewriting the workflow before transferring it into the target environment and uploading it to a workflow engine. In previous work (Weder et al., 2021b), we introduced an approach to automatically select a suitable quantum computer for the execution of quantum circuits during runtime. For this, the required services are automatically deployed, and the workflow is adapted to invoke them. In future work, we plan to integrate our analysis and rewrite method with this approach. However, conducting the method at workflow runtime requires generating the hybrid programs, which can result in impractical delays for time-critical use cases. Therefore, the user can decide if the rewrite during design or runtime is more suitable depending on the use case.

9 CONCLUSION & FUTURE WORK

Today‘s quantum applications are typically hybrid, comprising quantum and classical programs. Thus, the different programs have to be orchestrated and required data must be passed between them. By using workflow technologies to model these applications, one can benefit from their robustness, scalability, and automated error handling. Additionally, they can ease the understanding by providing a graphical notation. However, for the execution of hybrid loops, conducting quantum and classical programs multiple times, the orchestration using workflows is inefficient due to the increased latency and potential queuing times. Instead, hybrid runtimes can be used, deploying the quantum and classical programs closely together, and optimizing their communication. In this paper, we introduced a method to analyze quantum workflows to detect hybrid loops, that can benefit from hybrid runtimes. Furthermore, hybrid programs are automatically generated based on these loops, and the workflow is rewritten to invoke them instead of orchestrating the loops. To validate the practical feasibility of our approach, we presented a prototypical implementation, a case study showing an exemplary quantum workflow and how it is analyzed and rewritten, and an evaluation of the required time to conduct our method, as well as a runtime comparison of a workflow before and after applying our method.

In future work, we will evaluate how intermediate information about hybrid program executions can be extracted from the hybrid runtimes to enable live monitoring in the workflow engine using provenance views. Additionally, we want to extend the detection and filtering steps by taking into account additional non-functional requirements. Once new hybrid runtimes are offered, we plan to incorporate them into our presented analysis and rewrite framework. Finally, we will further evaluate our method, as well as the resulting speed-ups, using different workflow models.

ACKNOWLEDGEMENTS

The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in *Simulation Technology* (EXC 2075) at the University of Stuttgart. This work was partially funded by the BMWK project *PlanQK* (01MK20005N) and the project *SEQUOIA* funded by the Baden-Wuerttemberg Ministry of the Economy, Labour and Housing.

REFERENCES

Agrawal, R., Gunopulos, D., and Leymann, F. (1998). Mining Process Models from Workow Logs. In *International Conference on Extending Database Technology*, pages 467–483. Springer.

AWS (2021). Introducing Amazon Braket Hybrid Jobs – Set Up, Monitor, and Efficiently Run Hybrid Quantum-Classical Workloads. https://aws.amazon.com/de/blogs/aws/introducing-amazon-braket-hybrid-jobs-set-up-monitor-and-efficiently-run-hybrid-quantum-classical-workloads.

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

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC)*, pages 692–695. Springer.

Bucchiarone, A., Marconi, A., Pistore, M., and Raik, H. (2012). Dynamic Adaptation of Fragment-based and Context-aware Business Processes. In *Proceedings of the 19th International Conference on Web Services (ICWS)*, pages 33–41. IEEE.

Camunda (2021a). Camunda BPMN Modeler. https://camunda.com/products/camunda-bpm/modeler.

Camunda (2021b). Camunda BPMN Workflow Engine. https://camunda.com/products/camunda-bpm/bpmn-engine.

Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S. C., Endo, S., Fujii, K., et al. (2021). Variational Quantum Algorithms. *Nature Reviews Physics*, pages 1–20.

CNCF (2021). Kubernetes. https://kubernetes.io.

Cohen-Boulakia, S., Froidevaux, C., and Chen, J. (2012). Scientific Workflow Rewriting while Preserving Provenance. In *Proceedings of the 8th International Conference on E-Science*, pages 1–9. IEEE.

De, B. (2017). Api Management. In *API Management*, pages 15–28. Springer.

Di Penta, M., Esposito, R., Villani, M. L., Codato, R., Colombo, M., and Di Nitto, E. (2006). WS Binder: a Framework to enable Dynamic Binding of Composite Web Services. In *Proceedings of the 2006 international workshop on Service-oriented software engineering*, pages 74–80.

Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A. (2013). *Fundamentals of Business Process Management*, volume 1. Springer.

Eder, J. and Liebhart, W. (1996). Workflow Recovery. In *Proceedings of the International Conference on Cooperative Information Systems*, pages 124–134. IEEE.

Egger, D. J., Mareček, J., and Woerner, S. (2021). Warm-starting quantum optimization. *Quantum*, 5:479.

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

Farhi, E., Goldstone, J., and Gutmann, S. (2014). A Quantum Approximate Optimization Algorithm. *arXiv:1411.4028*.

Harrow, A. W., Hassidim, A., and Lloyd, S. (2009). Quantum Algorithm for Linear Systems of Equations. *Physical review letters*, 103(15):150502.

HashiCorp (2021). Terraform. https://www.terraform.io.

Havlíček, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., and Gambetta, J. M. (2019). Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212.

Herschel, M., Diestelkämper, R., and Ben Lahmar, H. (2017). A Survey on Provenance: What for? What Form? What from? *The VLDB Journal*, 26(6):881–906.

IBM (2021a). IBM Quantum delivers 120x speedup of quantum workloads with Qiskit Runtime. https://research.ibm.com/blog/120x-quantum-speedup.

IBM (2021b). IBM's roadmap for building an open quantum software ecosystem. https://research.ibm.com/blog/quantum-development-roadmap.

IBM (2021c). Qiskit Runtime. https://github.com/Qiskit-Partners/qiskit-runtime.

IBM (2021d). Qiskit Runtime Documentation. https://qiskit.org/documentation/partners/qiskit_runtime/tutorials/sample_vqe_program/qiskit_runtime_vqe_program.html.

Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J. M., and Gambetta, J. M. (2017). Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246.

Karalekas, P. J., Tezak, N. A., Peterson, E. C., Ryan, C. A., da Silva, M. P., and Smith, R. S. (2020). A quantum-classical cloud platform optimized for variational hybrid algorithms. *Quantum Science and Technology*, 5(2).

Képes, K., Breitenbücher, U., Sáez, S. G., Guth, J., Leymann, F., and Wieland, M. (2016). Situation-Aware Execution and Dynamic Adaptation of Traditional Workflow Models. In *Proceedings of the 5th European Conference on Service-Oriented and Cloud Computing (ESOCC)*, pages 69–83. Springer.

Khan, S. U., Awan, A. J., and Vall-Llosera, G. (2019). K-Means Clustering on Noisy Intermediate Scale Quantum Computers. *arXiv:1909.12183*.

Kopp, O., Binz, T., Breitenbücher, U., and Leymann, F. (2013). Winery – A Modeling Tool for TOSCA-based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC)*, pages 700–704. Springer.

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

LaRose, R. and Coyle, B. (2020). Robust data encodings for quantum classifiers. *Physical Review A*, 102(3):032420.

Leymann, F. (1995). Supporting Business Transactions via Partial Backward Recovery In Workflow Management Systems. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 51–70. Springer.

Leymann, F. and Barzen, J. (2020). The bitter truth about gate-based quantum algorithms in the NISQ era. *Quantum Science and Technology*, 5(4).

Leymann, F. and Barzen, J. (2021). Hybrid Quantum Applications Need Two Orchestrations in Su-

perposition: A Software Architecture Perspective. *arXiv:2103.04320*.

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

Leymann, F. and Roller, D. (2000a). Method and computer system for generating process management computer programs from process models. US Patent 6,011,917.

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

Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015). A Survey of Data-Intensive Scientific Workflow Management. *Journal of Grid Computing*, 13(4):457–493.

Maciejewski, F. B., Zimborás, Z., and Oszmaniec, M. (2020). Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography. *Quantum*, 4:257.

McCaskey, A. J., Dumitrescu, E. F., Liakh, D. I., and Humble, T. S. (2018). Hybrid Programming for Near-term Quantum Computing Systems. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–12. IEEE.

McClean, J. R., Romero, J., Babbush, R., and Aspuru-Guzik, A. (2016). The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2).

Müller, R., Greiner, U., and Rahm, E. (2004). AgentWork: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, 51(2):223–256.

Mundbrod, N., Grambow, G., Kolb, J., and Reichert, M. (2015). Context-Aware Process Injection: Enhancing Process Flexibility by Late Extension of Process Instances. In *On the Move to Meaningful Internet Systems (OTM)*, pages 127–145. Springer.

Nielsen, M. A. and Chuang, I. (2010). *Quantum Computation and Quantum Information*. AAPT.

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

OASIS (2013). *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*.

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

Orban, D. (2011). Templating and Automatic Code Generation for Performance with Python. *Cahier du GERAD G*, 2011:30.

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

Rinderle-Ma, S. and Reichert, M. (2010). Advanced Migration Strategies for Adaptive Process Management Systems. In *Proceedings of the 12th IEEE Conference on Commerce and Enterprise Computing*, pages 56–63. IEEE.

Salm, M., Barzen, J., Breitenbücher, U., Leymann, F., Weder, B., and Wild, K. (2020). The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms. In *Proceedings of the 14th Symposium and Summer School on Service-Oriented Computing (SummerSOC)*, pages 66–85. Springer.

Sim, S., Johnson, P. D., and Aspuru-Guzik, A. (2019). Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms. *Advanced Quantum Technologies*, 2(12):1900070.

University of Stuttgart (2021a). Demo Video. https://youtu.be/cXOkt0vVivo.

University of Stuttgart (2021b). Evaluation Data. https://github.com/UST-QuAntiL/qprov-content/tree/main/workflow-analysis-and-rewrite.

University of Stuttgart (2021c). QuantME Transformation Framework. https://github.com/UST-QuAntiL/QuantME-TransformationFramework.

University of Stuttgart (2021d). Quantum Workflow Use Cases. https://github.com/UST-QuAntiL/QuantME-UseCases.

Van Der Aalst, W. (2012). Process mining. *Communications of the ACM*, 55(8):76–83.

Vietz, D., Barzen, J., Leymann, F., Weder, B., and Yussupov, V. (2021). An Exploratory Study on the Challenges of Engineering Quantum Applications in the Cloud. In *Proceedings of the 2nd Quantum Software Engineering and Technology Workshop (Q-SET21)*, pages 1–12. CEUR Workshop Proceedings.

Waters, B. R., Balfanz, D., Durfee, G., and Smetters, D. K. (2004). Building an Encrypted and Searchable Audit Log. In *NDSS*, volume 4, pages 5–6. Citeseer.

Weder, B., Barzen, J., and Leymann, F. (2021a). MODULO: Modeling, Transformation, and Deployment of Quantum Workflows. In *Proceedings of the 25th International Enterprise Distributed Object Computing Workshop (EDOCW)*, pages 341–344. IEEE.

Weder, B., Barzen, J., Leymann, F., and Salm, M. (2021b). Automated Quantum Hardware Selection for Quantum Workflows. *Electronics*, 10(8).

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

Weder, B., Barzen, J., Leymann, F., and Vietz, D. (2022). Quantum Software Development Lifecycle. *Quantum Software Engineering*. arXiv:2106.09323.

Weder, B., Breitenbücher, U., Képes, K., Leymann, F., and Zimmermann, M. (2020a). Deployable Self-contained Workflow Models. In *Proceedings of the 8th European Conference on Service-Oriented and Cloud Computing (ESOCC)*, pages 85–96. Springer.

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

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

All links were last followed on February 23, 2022.