Institute of Architecture of Application Systems

Integrating Quantum Computing into Workflow Modeling and Execution

Benjamin Weder, Uwe Breitenbücher, Frank Leymann, and Karoline Wild

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{weder, breitenbuecher, leymann, wild}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{Weder2020_QuantumWorkflows,
  author    = {Benjamin Weder and Uwe Breitenb{\"u}cher and Frank Leymann and
               Karoline Wild},
  title     = {Integrating Quantum Computing into Workflow Modeling
               and Execution},
  booktitle = {Proceedings of the 13\textsuperscript{th} IEEE/ACM International
               Conference on Utility and Cloud Computing (UCC 2020)},
  year      = 2020,
  month     = dec,
  pages     = {279--291},
  doi       = {10.1109/UCC48980.2020.00046},
  publisher = {IEEE Computer Society}
}
```


Universität Stuttgart
Germany

Integrating Quantum Computing into Workflow Modeling and Execution

Benjamin Weder, Uwe Breitenbücher, Frank Leymann, and Karoline Wild
Institute of Architecture of Application Systems, University of Stuttgart, Germany
{*weder, breitenbuecher, leymann, wild*}*@iaas.uni-stuttgart.de*

***Abstract*—Quantum computing has the potential to significantly impact many application domains, as several quantum algorithms are promising to solve problems more efficiently than possible on classical computers. However, various complex pre- and post-processing tasks have to be performed when executing a quantum circuit, which require immense mathematical and technical knowledge. For example, calculations on today's quantum computers are noisy and require an error mitigation task after the execution. Hence, integrating classical applications with quantum circuits is a difficult challenge. In this paper, we introduce a modeling extension for imperative workflow languages to enable the integration of quantum computations and ease the orchestration of classical applications and quantum circuits. Further, we show how the extension can be mapped to native modeling constructs of extended workflow languages to retain the portability of the workflows. We validate the practical feasibility of our approach by applying our proposed extension to BPMN and introduce Quantum4BPMN.**

***Keywords*-Quantum Computing, Quantum Software, Quantum Applications, Workflow Technology, Modeling Extension**

I. Introduction

Quantum computing introduces a new computing paradigm and promises to solve many problems more efficiently than it is possible on classical computers [1], [2], [3]. Different *quantum algorithms* exist that provide a speed-up over their best-known classical counterparts [4], [5]. Examples of such quantum algorithms are *Grover's algorithm* [6] for unstructured search, *Shor's algorithm* [2] for factorizing numbers, or the *HHL algorithm* [7] for solving linear equations. In recent years, new or improved quantum computers were developed by hardware providers, such as IBM or Rigetti [8], [9]. Further, access to them is provided through *quantum cloud offerings*, such as *IBMQ*. Thus, quantum computers became publicly accessible, and use cases from various areas, such as computer science, chemistry, or physics, can be implemented and executed on real quantum computers [10].

However, the execution of a *quantum circuit*, i.e., an executable implementation of a quantum algorithm, requires additional pre- and post-processing tasks [9], [11]. These tasks are typically complex as they require immense mathematical and quantum-specific knowledge. For example, today's quantum computers do not allow to load arbitrary data into their registers, and, therefore, an initialization step has to be added to the beginning of the quantum circuit [11], [12]. This means the quantum circuit must be adapted depending on the input data, which requires mathematical knowledge about the implemented quantum algorithm as well as technical knowledge about the used quantum programming language [8]. Further, today's quantum computers are noisy and results of quantum computations are disturbed by some error [1]. Therefore, this error should be mitigated based on the error model of the used quantum computer, which again requires specific expertise about the quantum computer and suitable error mitigation techniques [9], [13].

Thus, integrating classical applications with quantum circuits is a difficult challenge, as these pre- and post-processing tasks have to be performed in order to properly execute the quantum circuit. Today, there exist several approaches to orchestrate the functionalities provided by different software artifacts [14]. *Workflow technology* is one of these orchestration approaches that has been proven to be applicable to integrate various heterogeneous types of applications [15]. Thereby, the required tasks, as well as their execution order and the data flow between them, are specified in so-called *workflow models*, which can then be automatically executed by a *workflow engine* [14], [15]. Due to its features, such as scalability, reliability, robustness, and transactional processing, workflow technologies provide a flexible orchestration approach to combine different kinds of applications. Hence, the technology seems to be promising to also integrate quantum circuits with classical applications of any kind. Moreover, the required pre- and post-processing tasks could also be included in workflow models, and therefore, automatically be executed by the workflow engine.

However, there is currently neither a means to model quantum circuit executions explicitly in workflow languages, such as *BPMN* [16], nor support existing workflow engines the invocation of quantum circuits. Technically, workflow technologies enable orchestrating quantum circuits with classical applications as arbitrary kinds of tasks can be modeled. However, the required pre-processing, execution, and post-processing tasks have specific characteristics, e.g., their input and output data or their configuration parameters [11]. Thus, this requires a lot of technical and mathematical knowledge, resulting in a time-consuming and error-prone modeling process, which can only be done by quantum experts.

In this paper, we (i) introduce a technology-independent modeling extension for imperative workflow languages called QuantME to model quantum computations in workflow models. This extension enables modeling the execution

of preconfigured quantum circuits while hiding the technical details, as well as the specification of the pre- and post-processing tasks to customize a quantum computation. However, as modeling extensions reduce the portability of the resulting workflow models, we (ii) present a method to transform workflow models using our modeling extension to workflow models containing only native modeling constructs of the used workflow language to ensure their portability. To validate the practical feasibility of our approach, we (iii) apply QUANTME to BPMN and introduce *Quantum4BPMN*. Finally, we (iv) present a prototypical implementation of the transformation method from Quantum4BPMN to BPMN and show a case study implementing three different quantum algorithms using our proposed modeling extension.

II. FUNDAMENTALS & PROBLEM STATEMENT

In this section, we introduce fundamentals about quantum computing and describe the typical process to develop and execute quantum circuits. Then, we discuss the basics of workflow technologies and present our problem statement.

A. Quantum Computing

In quantum computing, information is encoded within a *quantum system* [1], [5]. A *qubit*, which corresponds to a bit in classical computing, cannot only be in the states 0 and 1 but in both states at the same time, a so-called *superposition* [4], [5]. Multiple qubits can be combined into a *quantum register* [17]. Thus, a quantum register with n qubits can be in a superposition of 2^n states, represented by a 2^n-dimensional *state vector* in a complex vector space. Manipulations on the states of quantum registers are performed by *unitary transformations* [4]. Such manipulations modify the 2^n states of the register at the same time, and this *quantum parallelism* is one reason for the power of quantum computing [18]. In the gate-based quantum computing model, *quantum algorithms* define a set of unitary transformations that are applied to an input state of a quantum register to transform it into some output state [17]. The result of the quantum algorithm is then retrieved by performing a *measurement* on the quantum register [11]. The measurement returns a bit string representing one of the 2^n states of the quantum register [1], [4]. Quantum algorithms are inherently probabilistic, thus, they must be executed multiple times, leading to a probability distribution of results, and then the most frequent result is used [5], [17].

Different quantum algorithms exist that provide a speed-up over their best-known classical counterparts. For example, *Grover's algorithm* [6] for unstructured search, *Shor's algorithm* [2] for factorizing numbers, or the *HHL algorithm* [7] for solving linear equations. Another example is the *Quantum Approximate Optimization Algorithm (QAOA)* [19] that can be used for solving various optimization problems. Furthermore, different quantum hardware providers, such as IBM, Rigetti, or Honeywell, developed quantum computers in recent years and offer access to them, e.g., via the cloud [8], [17]. Thus, they became publicly accessible, and use cases from various areas, such as computer science, chemistry, or physics, can be implemented and executed on real quantum computers [10].

However, today's quantum computers are affected by noise from various sources, which can cause errors in computations [1], [13]. For example, unintended interactions of qubits with their environment lead to state changes over time, which is referred to as *decoherence* [4]. Further, the physical operations within a quantum computer cannot be executed perfectly [20]. Hence, the restricted capabilities of today's quantum computers have to be taken into account when developing and executing implementations of quantum algorithms. For example, faulty measurements can lead to a probability distribution of results of a computation that does not directly reflect the final state of the quantum computer. Therefore, the execution of a quantum computation should include a step to mitigate the influence of these errors [9].

There are different quantum computing models, e.g., adiabatic [21], gate-based [8], or measurement-based [22]. These models have been shown to be formally equivalent, but define quantum algorithms in different ways [21], [22]. In this work, we restrict our considerations to the gate-based quantum computing model as many available quantum computers are based on it [8]. However, our approach can be extended regarding other computing models in the future.

B. Development and Execution of Quantum Circuits

In the gate-based quantum computing model, a quantum algorithm is implemented as a so-called *quantum circuit* [1], [5], [23]. It consists of an input state, a set of *gates* operating on one or multiple qubits to manipulate the state, and a set of *measurements* to retrieve classical information from qubits [4], [17]. Thereby, quantum circuits can be defined, e.g., using quantum programming languages, such as *Q#*, quantum assembly languages, such as *OpenQASM* or *Quil*, or libraries that are embedded into non-quantum programming languages, such as *Qiskit* or *Forest* in Python [8]. In previous work, we analyzed the various phases a quantum circuit should go through and introduced the *quantum software lifecycle* [9]. Figure 1 presents a simplified development and execution process for quantum circuits.

First, in the *quantum circuit implementation* phase, the quantum circuit for the required quantum algorithm is implemented. This is done during development time. The quantum circuit is reusable for different problem instances that can be solved by the quantum algorithm. In this stage, the quantum circuit may also contain so-called *oracles* [24], which are used by some quantum algorithms, such as Grover [6]. An oracle is a subroutine realizing a hidden function, which is usually specific to the problem instance to solve. For example, Grover's algorithm for unstructured search uses an oracle that decides whether a given element is the

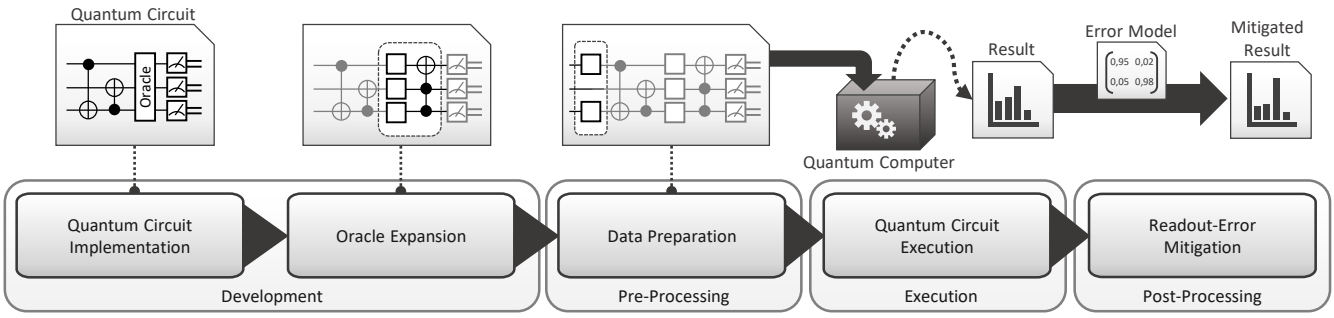

Figure 1. Simplified development and execution process of a quantum circuit

searched one or not. Therefore, the oracles of the quantum circuit have to be replaced by subcircuits implementing their functionality in an *oracle expansion* phase [11], [24]. Then, in the *data preparation* phase, the quantum computer has to be initialized with the input data for the specific problem instance to solve [9]. However, today's quantum computers only allow initializing their registers in the *all-zero state*, which means all qubits are set to zero [25]. Hence, the initialization must be done by prepending an appropriate subcircuit to the beginning of the original circuit, which prepares the required state [9]. This phase has to be done at execution time, as the input data is usually not known at development time. Therefore, the quantum circuit must be adapted during execution time depending on the input data, which requires immense technical knowledge. Afterward, the quantum circuit is executable and can be deployed and executed on a quantum computer. The result is a probability distribution produced when executing the quantum circuit multiple times (see Section II-A). However, this probability distribution is disturbed by so-called readout-errors [11], [26]. To reduce their influence on the result, in the last phase, *readout-error mitigation* should be applied based on the error model of the used quantum computer [9], [26].

C. Workflow Technology

Workflow technologies enable the modeling and execution of processes [14], [15]. Thereby, processes are specified in *workflow models* using a workflow language, such as the *Business Process Model and Notation (BPMN)* [16] or the *Business Process Execution Language (BPEL)* [27]. Workflow models consist of a set of *activities* that have to be performed to achieve a certain goal [28]. The functionality of these activities can be implemented in different ways. For example, an activity can invoke a web service, execute a script, or require a human action. The activities of a workflow model are connected by *control flow edges* that specify a partial order in which they are executed [15]. Further, *data flow* can be defined to transfer data between the activities of a workflow model [14]. Therefore, workflow technologies enable to orchestrate the functionalities provided by different software artifacts to achieve a certain goal [14], [15], [28].

One benefit of utilizing workflow technologies is the capability to automatically execute workflow models using a *workflow engine* [15], [28]. Thereby, the workflow engines provide a scalable and robust execution environment [14]. Additionally, by using a standardized workflow language, such as BPMN or BPEL, the portability of workflow models across different workflow engines can be achieved. Furthermore, many workflow languages and workflow engines implement comprehensive error handling mechanisms [15]. Hence, they, e.g., enable defining alternative control flows in the presence of failures [14], [28]. Additionally, transactions comprising multiple activities of a workflow model can be specified [15], [29]. Therefore, in the case of an error during the execution of the transaction, it can be automatically rolled back, or if that is not possible, specified compensation actions for the activities can be conducted [14], [15].

Due to these benefits, workflow technologies are essential for the modeling and execution of different kinds of processes, e.g., business processes but also scientific processes, such as complex simulations [14], [15], [30], [31]. With the growing capabilities of available quantum computers and the development of new quantum algorithms and corresponding quantum circuits, integrating quantum circuits in workflows to solve certain tasks is of main interest. However, there is currently neither a means to model quantum circuit executions explicitly in workflow languages, nor do existing workflow engines support the invocation of quantum circuits.

D. Problem Statement

As shown in Section II-B, several pre- and post-processing tasks, e.g., the data preparation or the oracle expansion, have to be performed when executing a quantum circuit. These tasks are typically complex and require mathematical knowledge as well as technical knowledge about the used programming language and quantum computer. Thus, integrating classical applications with quantum circuits is a significant challenge. Although workflow technologies provide the necessary basis for integrating heterogeneous applications, all phases of the quantum software lifecycle still need to be implemented as activities in workflow models. However, as there is currently no modeling support for these

activities abstracting from the technical and mathematical details, the modeling process is time-consuming, error-prone and can only be done by quantum experts. Moreover, the specification of each pre-processing, execution, and post-processing task as activity clutters the workflow model. Hence, an abstraction layer to model the execution of quantum circuits for which all these steps are preconfigured is required to hide the details and enable their orchestration by workflow modelers without deep knowledge in quantum computing. Thus, our first research question is as follows:

> *RQ 1: "What modeling extensions for workflow models are required to model the execution of quantum circuits, as well as the required pre- and post-processing tasks, to facilitate the orchestration of quantum circuits and classical applications?"*

However, by introducing a domain-specific modeling extension for quantum computing the resulting workflow models cannot be executed by existing workflow engines without extending them to enable the processing of the new modeling constructs. As we do not intend to develop a new workflow engine that supports the proposed extensions and aim to retain the portability of the workflow models, an approach to map the introduced modeling constructs to native modeling constructs of the used workflow language is needed. Thus, our second research question is as follows:

> *RQ 2: "How can the modeling extensions be mapped to native modeling constructs of existing workflow languages to retain the portability between different workflow engines?"*

III. Integrating Quantum Computing into Workflows

In this section, we introduce our method to integrate quantum computing into workflow modeling and execution. The method is depicted in Figure 2, which gives a high-level overview of our approach. In this method, we introduce the *Quantum Modeling Extension* (QuantME) for imperative workflow languages, which is discussed in detail in Section IV. QuantME supports modeling workflow models that execute quantum circuits following the lifecycle phases described in Section II-B (RQ1). Thus, it enables streamlining the pre-processing, execution, and post-processing tasks and integrating quantum circuits with classical applications.

In the first step, the workflow modeler creates a so-called QuantME *workflow model*, which is modeled using a workflow language fulfilling the requirements discussed in Section IV-A that has been extended by our proposed QuantME modeling extension. Thus, a QuantME workflow model contains native constructs of the used workflow language to support traditional activities, such as service invocations or the execution of scripts, as well as the

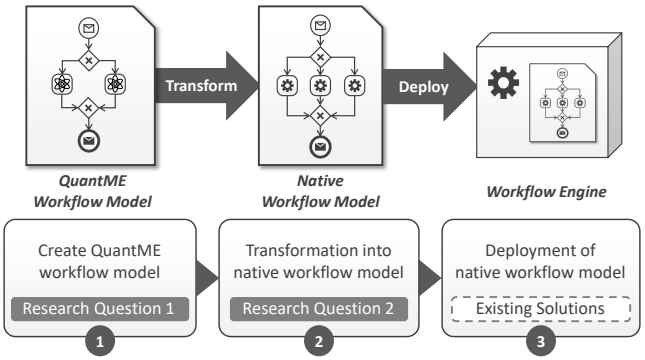

Figure 2. Overview of our integration method (based on [33])

QuantME modeling constructs. However, the introduced modeling extensions would reduce the portability of the QuantME workflow models, and we do not intend to develop a new workflow engine that supports the proposed extensions. Thus, in the second step, the QuantME workflow model is transformed into a *native workflow model* (RQ2). For this, we introduce an approach to automatically replace the QuantME modeling constructs by suited *workflow fragments* [32] from a repository, that implement the logic of the respective QuantME constructs. This replacement approach is presented in Section V. The resulting workflow models only contain natively supported modeling constructs of the used workflow language. Hence, our approach allows to benefit from the abstractions of the introduced modeling extension while retaining the portability of the workflow models across workflow engines supporting the same workflow language. In the last step, the native workflow model is deployed to a workflow engine and can be instantiated. To validate the practical feasibility of our approach, we introduce Quantum4BPMN in Section VI-A, which is a modeling extension for BPMN that supports QuantME.

IV. QuantME: A Modeling Extension for Quantum Computing

In the following, we first discuss requirements that have to be fulfilled by a workflow language to be extendable by QuantME. Then, we introduce the QuantME modeling constructs, discuss their purpose and their semantics.

A. Requirements on Workflow Languages

To be extendable by QuantME the target workflow language has to satisfy some requirements, which we discuss in this subsection. First, (i) the workflow language must support the notion of an activity or task to define a single execution step in the modeled process. Furthermore, (ii) these activities have to allow the definition of attributes to configure them depending on the context where they are used in the workflow. The workflow language must also (iii) enable the specification of control flow between different activities to define a partial order in which they

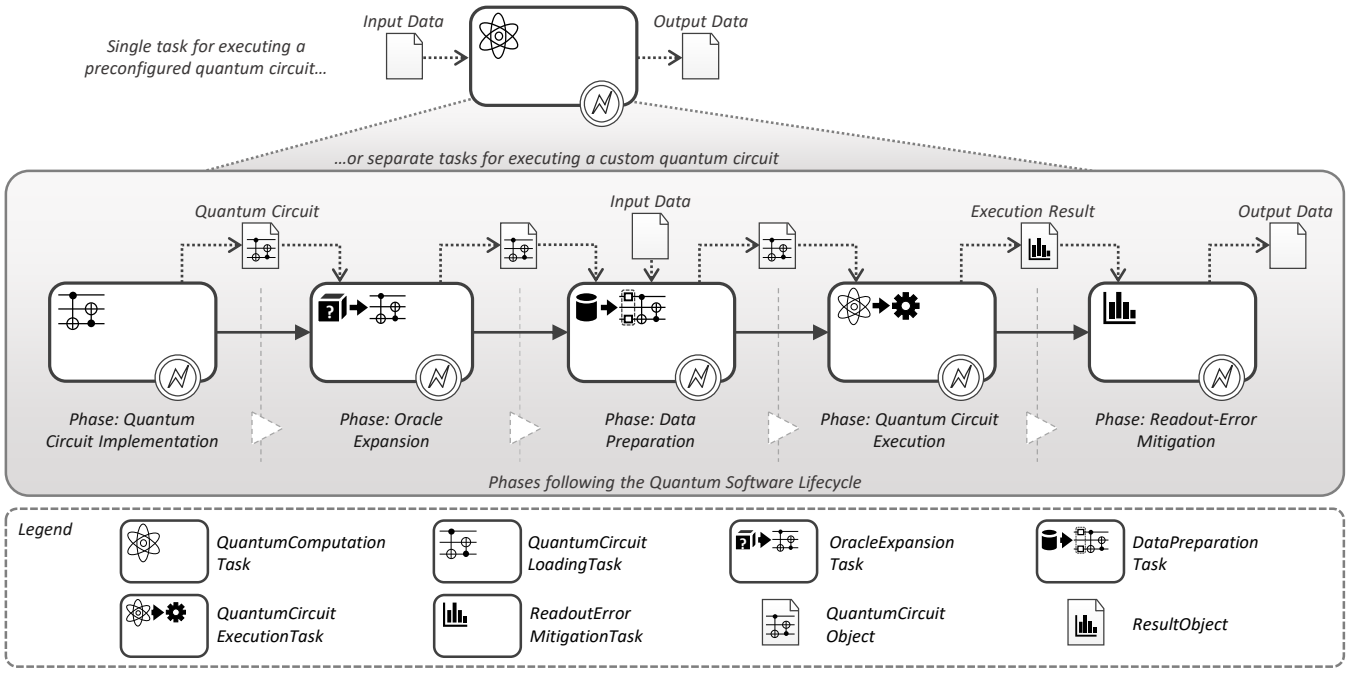

Figure 3. Overview of the QuantME modeling constructs

are executed. Additionally, (iv) it must be possible to model the transfer of data between the activities of a workflow model. Finally, (v) the workflow language has to support to handle exceptions by defining alternative control flows in the presence of errors. These requirements are, e.g., satisfied by BPMN [16], which is used in many domains, provides a well-known graphical notation, and is supported by state-of-the-art workflow engines, such as the *Camunda BPMN engine* [34]. Therefore, we use the BPMN concepts and their graphical notation to describe our modeling extension. However, the requirements are also fulfilled by other workflow languages, and Section VII discusses by which BPEL [27] modeling constructs the requirements are satisfied.

B. Executing Custom Quantum Circuits

To execute a quantum circuit from within a workflow model, tasks implementing all the phases of the quantum software lifecycle presented in Section II-B have to be specified in the workflow model. First, the quantum circuit has to be loaded into the workflow model and the oracles of the circuit must be replaced with corresponding implementations. Then, the circuit has to be initialized with the input data. Subsequently, it can be deployed and executed on a quantum computer. Finally, readout-error mitigation is used to reduce the influence of errors in the results. Therefore, several complex tasks have to be modeled to execute a quantum circuit.

In the following, we introduce a new QUANTME modeling construct for each lifecycle phase, as depicted in Figure 3. Furthermore, a set of configuration parameters is defined for the modeling constructs, which are important to customize the corresponding lifecycle phase. Thus, the introduced modeling constructs guide and support quantum experts in modeling invocations of custom quantum circuits while enabling them to configure the pre-processing, execution, and post-processing tasks to their needs. Thereby, each of the modeling constructs allows defining alternative control flows in the case of an error, e.g., if the execution of a quantum circuit fails. The possibility to define such alternative control flows is one requirement for workflow languages that are extensible by QUANTME and can, e.g., be achieved by an error boundary event in BPMN. However, the use of events is not limited to error handling. If further events are natively supported by the used workflow language, such as the timer event in BPMN, they can also be utilized.

1) Transferring Data about a Quantum Computation: The execution of the different pre-processing, execution, and post-processing tasks defined in the quantum software lifecycle, requires to transfer data between them. For example, the code of the quantum circuit has to be exchanged between multiple tasks to modify it based on the input data or to execute it. Although the required data can also be transferred using general data objects, a new kind of data object clearly defines the relevant attributes, and hence, eases the modeling process. Thus, we introduce a new `QuantumCircuitObject`, which is an extension of the general data object for storing and transferring all relevant data about a quantum circuit. It defines two attributes: (i) `QuantumCircuit` that contains the code representing the quantum circuit and (ii) `ProgrammingLanguage`,

which defines the programming language that was used to implement the quantum circuit (see Section II-B) to enable the proper interpretation of the code. Further, a second data object called `ResultObject` is introduced to transfer the results of quantum computations. It defines one attribute: *ExecutionResult* that is used to store the probability distribution of different results from a quantum circuit execution.

2) Loading a Quantum Circuit: For executing a quantum circuit, first, the code of the circuit has to be loaded into the workflow model. Therefore, we introduce a new task type called `QuantumCircuitLoadingTask`. Its semantics is the creation of a `QuantumCircuitObject` containing the code of the quantum circuit and all related data. Other QUANTME tasks in a workflow model can then use the `QuantumCircuitObject`, e.g., to modify the quantum circuit or to execute it (see Figure 3). The task defines two attributes: (i) an optional `QuantumCircuit`, which can be used to directly insert the code of the quantum circuit into the workflow model, and (ii) an optional `URL` that allows specifying a location to load the circuit code from. However, one of the two optional attributes has to be set to enable loading the quantum circuit into the workflow model.

3) Expanding Oracles: The loaded quantum circuit from the previous phase may contain some oracles, which have to be replaced by subcircuits implementing their functionality [11], [24]. Therefore, we introduce a new `OracleExpansionTask`, which has the semantics to replace an oracle of the given quantum circuit by a problem specific subcircuit. The task uses two `QuantumCircuitObjects` containing the input and output quantum circuits (see Figure 3). Thereby, in the output quantum circuit, the specified oracle is replaced. If the circuit contains further oracles, they can be replaced by additional `OracleExpansionTasks`. The task has three attributes: (i) `OracleId` identifying the oracle to expand in the quantum circuit, (ii) an optional `OracleCircuit` that can be used to directly specify the subcircuit to replace the oracle, and (iii) an optional `OracleURL` which allows loading the oracle, e.g., from a repository. Thereby, the `OracleId` can be the id of an object in the quantum circuit if the programming language explicitly supports defining oracles or the position in the quantum circuit where the oracle has to be inserted otherwise. Furthermore, either the optional `OracleCircuit` or the `OracleURL` attributes have to be specified to successfully replace the oracle.

4) Preparing the Input State: After loading the quantum circuit into the workflow model and expanding contained oracles, the input data has to be encoded into the quantum circuit [9], [11]. To model this, we introduce a new task type called `DataPreparationTask`. Its semantics is the addition of a subcircuit to the beginning of the original circuit to prepare the required state in the register of the quantum computer based on the given input data [12], [25]. For this, the quantum circuit to initialize is passed to it through a `QuantumCircuitObject` and the input data through another data object, as shown in Figure 3. Then, the output of the task is a new `QuantumCircuitObject` with the initialized circuit. The task has two attributes: (i) `EncodingScheme` defining how to encode the input data into the initializing circuit and (ii) `ProgrammingLanguage` specifying the programming language of the quantum circuit that needs to be initialized by the task (see Section II-B). Thereby, different quantum algorithms often require the same encoding scheme, such as *basis* or *analog encoding* [11], [12], and thus, the workflow modeler can define the kind of encoding that is required for his quantum circuit using the `EncodingScheme` attribute.

5) Executing a Quantum Circuit: After preparing the input data, the quantum circuit can be deployed and executed on a quantum computer in the next phase. For this, a new task type called `QuantumCircuitExecutionTask` is introduced. It has the semantics to execute the circuit passed to it by a `QuantumCircuitObject` on a specified or automatically selected quantum computer and to return a `ResultObject` containing the probability distribution of results produced by the execution (see Figure 3). The task has three attributes: (i) an optional `Provider` that can be used to define the quantum cloud offering for the execution, (ii) an optional `QPU` that enables specifying a concrete quantum computer to use, and (iii) an optional `Shots` to define the number of executions. The specification of a `Provider` allows selecting the quantum cloud offering based on available credentials or other criteria, such as the incurred costs. Furthermore, the `QPU` attribute enables to exactly define the quantum computer for the execution. If it is not specified by the modeler, a suitable quantum computer is automatically selected while restricting the selection to quantum computers of the `Provider` if defined [35].

6) Mitigating the Readout-Error: The execution of a quantum circuit results in a probability distribution of results from multiple runs, which differs from the ideal distribution that should result from the measurements of the output state of the quantum computer due to readout-errors [9], [11], [36]. To reduce the influence of these errors, a new `ReadoutErrorMitigationTask` is introduced. It has the semantics to mitigate the error in a probability distribution of results from a quantum computation which is passed to it by a `ResultObject`. The task defines three attributes: (i) `UnfoldingTechnique` specifying the unfolding technique to use to mitigate the error, (ii) `QPU` identifying the quantum computer that was used for the execution, and (iii) an optional `MaxAge` to define a maximum age of the required data to use for the mitigation. Thereby, different unfolding techniques exist, such as the *iterative dynamically stabilized unfolding method* [37] or the *correction matrix unfolding technique* [26], which can lead to different qualities of error mitigation for various quantum circuits and quantum computers. Hence, the workflow modeler can specify which

technique to use in the `UnfoldingTechnique` attribute. These techniques rely on collected provenance data about the used quantum computer, which can change significantly over time [13]. Therefore, the specification of a small `MaxAge` can increase the quality of the mitigation. However, the collection of the provenance data requires the execution of lots of quantum circuits on the quantum computer and can incur high costs [26]. Hence, the workflow modeler can address this trade-off by defining a suitable `MaxAge` for his use case. The result of the `ReadoutErrorMitigationTask` is stored in a data object and can be used by classical applications, e.g., modeled as service tasks, that require the result of the quantum circuit or perform some post-processing.

C. Executing Preconfigured Quantum Circuits

The introduced data objects, as well as the new task types for the pre-processing, execution, and post-processing phases, are intended for quantum experts who want to customize the execution of arbitrary quantum circuits. However, to execute a quantum circuit for which all these steps are already preconfigured, we present a new task type that combines these tasks in a hidden manner only exposing one abstract task to the workflow modeler. Thus, it can be used by modelers without expertise in quantum computing who can reuse the circuits defined by quantum experts to integrate them with other heterogeneous applications in workflows.

The newly introduced task type has the visual representation shown on top of Figure 3 and is called `QuantumComputationTask`. Its semantics is the execution of a certain quantum algorithm implemented by the quantum circuit on the input data that is passed to it through a data object and to store the results of the algorithm in another data object. The task has two attributes: (i) `Algorithm` that specifies the quantum algorithm to execute and (ii) an optional `Provider` defining the quantum cloud offering that should be used for the execution of the quantum algorithm. Thereby, the `Algorithm` attribute specifies the id of a workflow fragment containing all the pre-processing, execution, and post-processing tasks to execute a quantum circuit implementing this algorithm. Thus, such workflow fragments can be stored by quantum experts in a repository to enable their reusability and used to replace the `QuantumComputationTask` while transforming the workflow model to a native workflow model (see Section V). In addition to the workflow fragments, the repository also contains a specification of their input and output data format to ease the integration with other heterogeneous applications. The definition of a `Provider` enables the workflow modeler to select the quantum cloud offering based on criteria, such as available credentials or incurred monetary costs. Therefore, a quantum computer of the specified quantum cloud offering is selected for the execution. If the attribute is not set, the quantum cloud offering, as well as the concrete quantum computer, can be automatically chosen [35].

V. Transformation to Native Workflow Models

In the following, we show how to transform a QuantME workflow model to a native workflow model to ensure its portability between different workflow engines supporting the used workflow language. Thus, the transformation method corresponds to step 2 of our integration approach.

A. Transformation Method

To transform QuantME workflow models to native workflow models, we introduce a (semi-)automatic transformation method, which consists of three steps. In the first step, the QuantME workflow model is specified using our modeling extension. However, the QuantME modeling constructs reduce the portability of the workflow model, and therefore, are automatically replaced in the second step. Hence, we introduce the so-called QuantME *Replacement Models (QRM)*, which we discuss in detail in the next subsection. QRMs define a replacement of QuantME tasks by suited workflow fragments that implement the required functionality. Therefore, the QuantME modeling constructs in a workflow model are iteratively selected and replaced until a native workflow model is obtained. If the QuantME workflow model contains a QuantME task for which no suited QRM is available, the transformation is aborted, and the user is informed. After the successful transformation, the native workflow model can be manually refined by the user in the last, optional step of the method.

B. QuantME Replacement Models

In the following, we define the structure of QuantME *Replacement Models (QRMs)*, which are the basis of our transformation method. Furthermore, we show how to decide if a QRM can be used to replace a QuantME task. Finally, the mapping of the control and data flow when replacing QuantME tasks by suited workflow fragments is discussed.

An exemplary QRM is depicted in Figure 4. QRMs consist of two parts: (i) a *detector* that specifies for which QuantME tasks the QRM can be applied and (ii) a *replacement fragment* defining the workflow fragment that implements the required functionality to replace the QuantME task in the detector. Thereby, the detector defines exactly one QuantME task type and a set of values for the attributes of this task type. For the attributes, detectors can define exactly one value, a list of possible values, or a wildcard (asterisk as value). Therefore, a QuantME task matches a detector, if it has the same task type and for all attributes either (i) the same value, (ii) one of the values in the specified list of possible values, or (iii) an arbitrary value if a wildcard is defined in the detector. For example, the QRM depicted in Figure 4 can be used to replace `ReadoutErrorMitigationTasks`, which require to apply the *correction matrix* unfolding technique for quantum computations executed either on *ibmq_rome* or *ibmq_london* and with an arbitrary value for the `MaxAge` attribute.

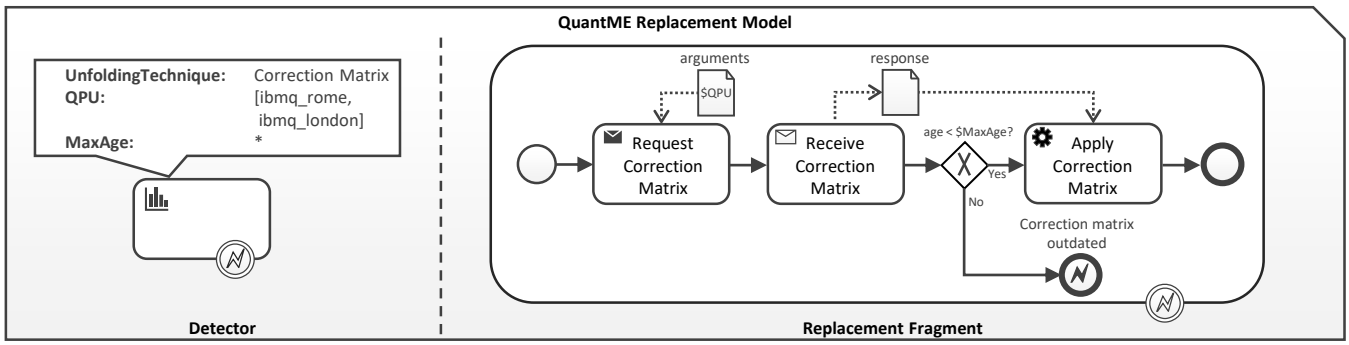

Figure 4. Exemplary QuantME Replacement Model (QRM)

If a matching detector for a QUANTME task is found, the replacement fragment of the QRM can be used to substitute the task in the workflow model. Thereby, the control and data flow have to be adapted. If the replacement fragment contains only one task, the control flow can be adapted by a static algorithm, that redirects in- and out-going control flow of the replaced task to the task of the replacement fragment. This can, e.g., be achieved in BPMN by implementing the replacement fragment as a *sub-process* or in BPEL by using a *scope*. In the same way, in- and out-going data flow can be adapted by attaching it to the replacing task in this case. If more than one task is contained in the replacement fragment, explicit mappings can be specified as proposed by Harzenetter et al. [38] for deployment models, that define how control and data flow of different kinds have to be handled during replacement. Furthermore, attributes of QUANTME tasks can be referenced from within replacement fragments to configure them depending on the attribute values of the replaced tasks. For example, in Figure 4, the value of the `QPU` attribute of the replaced QUANTME task is used to request the correction matrix of the used QPU. Thus, the references are substituted by the corresponding values of the QUANTME task during transformation.

QRMs are also allowed to contain QUANTME tasks in the replacement fragment, which are then replaced in subsequent iterations of the transformation method. Therefore, quantum experts can use the introduced modeling extensions to model replacement fragments for different `QuantumComputationTasks`, as well as reusable implementations of the various tasks for the lifecycle phases, e.g., for a certain data encoding or an unfolding technique.

VI. PROTOTYPICAL VALIDATION

In this section, we prove the practical feasibility of our approach. Thus, we first show how QUANTME can be applied to BPMN and introduce our prototype that implements the transformation to native BPMN. Afterward, we present a case study implementing three quantum algorithms using our modeling extensions and an evaluation of the modeling effort when using QUANTME compared to native BPMN.

A. Quantum4BPMN

To enable the practical application of our approach, we introduce an extension of BPMN [16] that supports QUANTME, called *Quantum4BPMN*. As we used the BPMN concepts and their graphical notation to define the QUANTME modeling constructs, the required extensions can easily be performed. For this, we added six new task types extending the general BPMN *task* construct with the required attributes of the QUANTME task types. Further, we extended the BPMN *data object* to fit the semantics of the introduced data objects. The proposed extension is accessible on Github[1].

B. Prototype

In the following, we introduce our prototype implementing the transformation method described in Section V for Quantum4BPMN. For this, we extended the *Camunda Modeler*, which is an open-source BPMN modeling framework. We added a plugin to enable modeling BPMN workflow models using Quantum4BPMN modeling constructs or loading them in XML format into the Camunda Modeler. Furthermore, a repository for QUANTME Replacement Models was added, which facilitates storing pairs of BPMN workflow models representing the detector and replacement fragment. Additionally, the transformation logic was implemented, which uses the replacement models from the repository to iteratively replace the contained QUANTME modeling constructs and returns native BPMN workflow models. The resulting workflow models can then be visualized and further modified manually in the Camunda Modeler. Finally, they can be exported for execution on a BPMN engine. The prototype is publicly available as an open-source project on Github[2].

C. Case Study

In this section, we demonstrate the usage of QUANTME by presenting three workflow models implementing *Simon's algorithm* [39], the *Bernstein-Vazirani algorithm* [40], [41], and *Grover's algorithm* [6]. Furthermore, to illustrate the

[1]https://github.com/UST-QuAntiL/QuantME-Quantum4BPMN
[2]https://github.com/UST-QuAntiL/QuantME-TransformationFramework

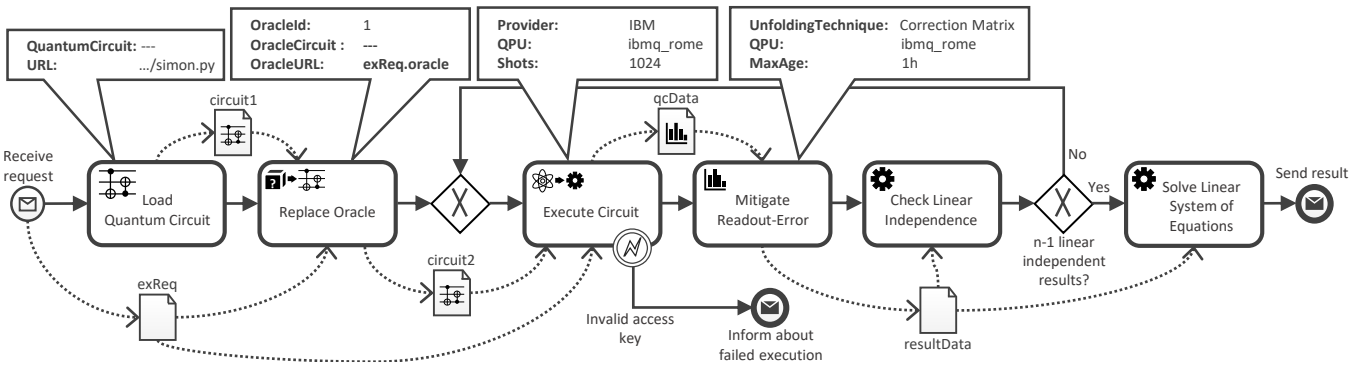

Figure 5. A scenario showing a potential implementation of Simon's algorithm using the QuantME modeling constructs

proposed transformation method, a transformed version of the workflow models is accessible on Github[3]. Therefore, they can be used to execute the three quantum algorithms by a BPMN engine, e.g., the Camunda engine [34].

1) Simon's Algorithm: In Figure 5 an overview of a workflow model implementing Simon's algorithm [39] is shown. Given a function $f(x) : \{0,1\}^n \rightarrow \{0,1\}^n$, the purpose of Simon's algorithm is to determine whether the function is bijective, i.e., it performs a one-to-one mapping between the elements of the source and target sets, or if it is a two-to-one function always mapping two source elements to the same target element. In the case of a two-to-one function, there exists a secret bit string s, which, if applied to an arbitrary source element using xor, leads to the second element that is mapped to the same target element. Thus, the goal is to determine if f is bijective and if not to retrieve the secret bit string s. Using a classical algorithm, up to $2^{n-1}+1$ evaluations of the function are needed to solve the problem [39]. In contrast, Simon's algorithm only requires $O(n)$ evaluations, leading to an exponential speed-up.

The workflow is initiated when a user sends an execution request (*exReq*) containing the URL to the oracle implementing the function f to investigate and the access key for the quantum cloud offering to use. First, the quantum circuit is loaded from the specified URL and stored in *circuit1*. The oracle in the circuit is replaced by the implementation from the given URL in the next step, and the resulting circuit is stored in *circuit2*. Then, the circuit is executed on the *ibmq_rome* quantum computer, which is accessible through IBM's cloud offering IBMQ. If an invalid access key is provided, the workflow terminates and the user is informed. Otherwise, the readout-error in the result is mitigated using the correction matrix unfolding technique. Next, a service task checks how many of the already received results are linearly independent, and if less than $n-1$, the circuit is executed again. After receiving $n-1$ linearly independent results, the linear system of equations is solved, leading to the searched bit string s, which is sent back to the user.

[3]https://github.com/UST-QuAntiL/QuantME-UseCases

2) Bernstein-Vazirani Algorithm: The Bernstein-Vazirani algorithm [40], [41] solves the problem of finding a hidden bit string that is encoded in a function. Thereby, the function $g(x) : \{0,1\}^m \rightarrow \{0,1\}$ returns the scalar product of the input bit string and the hidden bit string modulo 2. The best-known classical algorithm for this problem requires m evaluations of g for a bit string of size m. However, using the Bernstein-Vazirani algorithm, only one evaluation is needed.

A workflow model implementing the Bernstein-Vazirani algorithm is presented in Figure 6. Thereby, the configuration attributes for the four contained tasks are displayed in the light gray, as well as white boxes above the tasks. In contrast to Simon's algorithm, no classical post-processing is required for this algorithm, and thus, further conventional BPMN service tasks are not needed. The URL to load the quantum circuit is changed to load a circuit implementing the Bernstein-Vazirani algorithm. Further, the circuit should be executed on the *ibmq_athens* quantum computer instead of *ibmq_rome* and with only 512 shots. However, as we implemented a generic QRM for all quantum computers available over IBMQ and with a configurable shots parameter, no new QRM is needed to transform the shown QUANTME workflow model into a native workflow model.

3) Grover's Algorithm: The purpose of Grover's algorithm [6] is to search an item in an unsorted list of N items. For this, an oracle that checks whether a given item is the searched one is used. A quantum circuit implementing the algorithm consists of multiple iterations, which is referred to as *Grover iteration*. Each Grover iteration includes the execution of the oracle and so-called *amplitude amplification* [42]. Thereby, $\frac{\pi}{4} \times \sqrt{N}$ iterations are required [6]. Thus, Grover's algorithm can find the searched item with $O(\sqrt{N})$ oracle executions. In contrast, classical unstructured search has a runtime of $O(N)$, leading to a quadratic speed-up [6].

For the execution of Grover's algorithm using QUANTME, the same tasks as for the Bernstein-Vazirani algorithm can be used, and only the configuration through the corresponding attributes has to be changed. The workflow model is depicted in Figure 6, whereby the

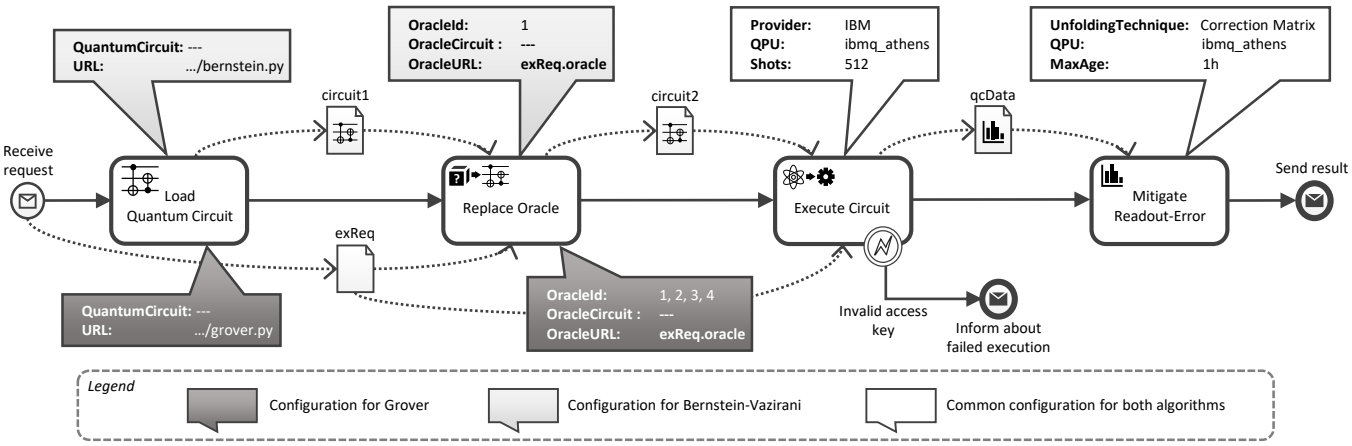

Figure 6. Workflow model implementing the Bernstein-Vazirani or Grover algorithm depending on the task configuration

configuration for the first two tasks is displayed in dark gray. For the `QuantumCircuitLoadingTask`, the URL is changed to a location of a quantum circuit implementing Grover's algorithm. The modeled workflow uses the *ibmq_athens* quantum computer with five qubits, and thus, the maximum value for N is $2^5 = 32$. According to the formula introduced before, four Grover iterations have to be executed. Therefore, the quantum circuit that is loaded in the first task contains four oracles that must all be replaced by the oracle implementation from the execution request. Thus, the `OracleExpansionTask` has four IDs defined and replaces all placeholders in the initial quantum circuit by the oracle provided at the specified URL. Alternatively, also multiple `OracleExpansionTasks` could be used, each replacing one of the oracles in the quantum circuit.

D. Evaluation

In the following, we present the results of our evaluation regarding the reduced modeling effort when modeling quantum algorithms in workflows using QUANTME, as well as the reusability of QRMs for workflow models that execute different quantum algorithms based on our case study.

Table I compares the number of required workflow tasks when integrating the different quantum algorithms without using our modeling extension to the number of tasks when using QUANTME. Thereby, when exploiting the `QuantumComputationTask`, only one task has to be modeled to execute the quantum algorithms from within the workflow models. If the QUANTME tasks for the various lifecycle phases are utilized (see Section IV-B), six and four tasks have to be modeled for the quantum algorithms, as shown in Figure 5 and Figure 6. In contrast, when using only conventional BPMN tasks, nine tasks are needed to implement Simon's algorithm and seven tasks for the Bernstein-Vazirani algorithm and Grover's algorithm respectively.

Furthermore, we evaluated the reuse of the created QRMs when modeling the three quantum algorithms. After implementing Simon's algorithm, no new QRM for the lifecycle phases had to be specified for the two other quantum algorithms. Both algorithms rely on the same tasks, and the detectors of the QRMs that were created for Simon's algorithm match the attributes for the two other algorithms (see Section V-B). For example, we use quantum computers available over IBMQ for all quantum algorithms, and, thus, the QRM for the `QuantumCircuitExecutionTask` can be reused, as it is capable of executing quantum circuits on any quantum computer from IBMQ. Hence, we demonstrated that defined once this expert knowledge can be reused by several workflow models implementing different quantum algorithms. As a result, if another quantum cloud offering should be used instead of IBMQ, only one new QRM has to be specified that can be utilized by all three workflow models. However, the Bernstein-Vazirani and Grover's algorithm are comparatively simple quantum algorithms that do not require further pre- or post-processing. For more complex algorithms, e.g., Shor's algorithm [2], additional tasks have to be implemented and added to the workflow models. Therefore, in future work, we plan to focus especially on variational algorithms [43], such as VQE [44] or QAOA [19], as they can be used to already benefit from quantum computing, even with today's restricted devices.

Table I
MODELING CONSTRUCT REDUCTION BY QUANTME

Quantum Algorithm	# Tasks using QCT*	# Tasks using QUANTME Lifecycle Tasks	# Tasks without QUANTME
Simon	1	6	9
Bernstein-Vazirani	1	4	7
Grover	1	4	7

* QCT = QuantumComputationTask

VII. Discussion

In this section, we first discuss the generality of QuantME. Then, possible application areas for workflows executing quantum circuits are sketched, and it is described what additional tasks may be required to execute them in these areas.

For a workflow language to be extendable by QuantME, some requirements have to be satisfied, as described in Section IV-A. In the following, we show how these requirements are met by the workflow language BPEL [27]: It supports (i) the notion of *activities* to describe single execution steps in a workflow and enables (ii) their configuration by *attributes*. Furthermore, (iii) control flow can, e.g, be specified by a *flow* activity. In contrast to data objects in BPMN, BPEL supports (iv) passing data between activities using *variables*. Hence, in BPEL, the different QuantME tasks have to handle their input and output data by variables. Finally, it (v) enables defining alternative control flows in the presence of errors using *fault handlers*. Thus, BPEL can be extended by QuantME, especially because BPMN can be transformed into BPEL, as discussed by several works [16], [45], [46], and therefore, also Quantum4BPMN. In future work, we will further evaluate the generality of QuantME by investigating mappings to other workflow languages.

An application area, where quantum computing is expected to enable breakthroughs in the future is machine learning [47]. Therefore, the research area of *quantum machine learning (QML)* attracts a lot of attention at the moment [48]. With the integration of quantum computing, workflows can help to develop, model, and execute new QML solutions by automating the different required steps. Another promising application area is scientific simulations, as quantum computing enables to simulate complex physical systems, that cannot be simulated on classical hardware [1]. However, in both application areas, huge amounts of data have to be processed to retrieve useful results. Hence, additional tasks have to be integrated into workflow models, e.g., loading the required data, preparing it for further processing, selecting a suited subset of the data, or visualizing the results after the quantum computation. Thereby, solutions from the research area of scientific workflows can be utilized to model and execute these tasks [30], [49]. In future work, we plan to integrate these solutions into our approach to ease the usage of workflows executing quantum circuits in these areas.

VIII. Related Work

Recently, Zapata announced *Orquestra* [50], a software platform for executing so-called *quantum-enabled workflows*. Orquestra provides a YAML-based workflow language to define workflows comprising tasks that are executed on classical hardware, as well as on quantum computers from different providers, such as IBM, Rigetti, and Atos. The tasks that can be modeled in an Orquestra workflow are defined and implemented by so-called *resources*. However, there exists no resource for each lifecycle phase. Instead, the user has to implement the required phases by himself. Additionally, there currently exists neither a graphical notation for the different resources nor a graphical modeling tool. Thus, the development of workflows is complex and requires a lot of technical expertise. In contrast, our approach provides a graphical notation and builds upon modeling constructs that are well-known to workflow modelers, reducing the modeling complexity. Furthermore, our approach enables benefiting from the variety of features provided by state-of-the-art workflow engines, such as transactional processing, robustness, scalability, or support for human tasks.

Various tools support one or multiple phases of the quantum software lifecycle, and thus, can be used when creating QRMs. McCaskey et al. [36] present the *eXtreme-scale ACCelerator (XACC)*, a hardware-agnostic execution framework for quantum computing. Thereby, they specify a compilation and execution process that enables conducting quantum circuits on various quantum computers independent of the used language. Further, they integrate the readout-error mitigation phase into this process. Thus, when using XACC, this phase does not need to be modeled explicitly. However, details about the mitigation, such as the used unfolding method, are hidden in the request to XACC then. Häner et al. [51] propose a methodology to optimize quantum circuits, which can be applied before their execution to reduce the influence of errors. Salm et al. [35] outline an approach to select a suited quantum computer for the execution of a quantum algorithm on a certain input and take into account, e.g., the number of available and required qubits. Thus, the approach can be used if the workflow modeler does not explicitly define a quantum computer.

Different research works propose domain-specific extensions for workflow languages and show a mapping to native modeling constructs of the used workflow language. For example, Falazi et al. [52] introduce a modeling extension to integrate blockchain interactions into workflow models and provide a transformation of their modeling constructs to native BPMN workflow fragments. Breitenbücher et al. [33] introduce an extension to model situations in workflows and present a mapping to native BPEL modeling constructs.

Several works utilize reusable workflow fragments to assemble an overall workflow. Eberle et al. [32] introduce a workflow fragment repository and a composer to create new workflows from existing workflow fragments. Sethi et al. [53] analyze the potential of reusable workflow fragments in different domains. Bucchiarone et al. [54] present an approach to dynamically adapt workflow models by replacing abstract activities with fine-grained workflow fragments based on defined goals. Képes et al. [55] describe an approach to dynamically select a workflow fragment for a required operation depending on the current situation. Schumm et al. [56] introduce the *Fragmento* repository to store workflow fragments providing functions such as validation and query support to find a workflow fragment

for a certain purpose. Garijo et al. [57] present a concept to detect common workflow fragments in scientific workflows based on collected provenance data and artificial intelligence. Wen et al. [58] describe a mechanism to discover suited workflow fragments depending on user requirements.

IX. Conclusion and Future Work

Quantum computing promises enormous speed-ups in solving many problems compared to classical algorithms. However, different complex pre- and post-processing tasks have to be performed when executing a quantum circuit, which require immense mathematical and technical knowledge. Therefore, the integration of classical applications and quantum circuits is a difficult challenge, as the classical application has to perform all these tasks to interact with the quantum circuit. In this work, we introduced the *Quantum Modeling Extension (*QUANTME*)* to model quantum circuit invocations in workflows to ease their orchestration with classical applications and showed how to ensure the executability of QUANTME workflow models on different workflow engines. We validated the technical feasibility of our approach by a prototypical implementation and a case study modeling three different quantum algorithms as workflow models using Quantum4BPMN and provided a first evaluation of the achieved reuse and degree of simplification.

In future work, we plan to develop a unified API for different quantum cloud offerings to enable executing quantum circuits on various offerings without requiring to implement special replacement fragments. Additionally, we will further evaluate our approach by implementing additional quantum workflows using our modeling extension and develop corresponding QRMs. Thereby, we focus especially on variational algorithms and analyze if extensions to QUANTME are needed to support the modeler in defining their special structure and tasks, such as evaluating cost functions or optimizing parameterized gates. Finally, we plan to incorporate solutions from the research area of scientific workflow into our approach, e.g., to handle large amounts of data.

Acknowledgment

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 – 390740016) at the University of Stuttgart. This work was partially funded by the BMWi project *PlanQK* (01MK20005N) and by the DFG project *DiStOPT* (252975529).

References

[1] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[2] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM Journal on Computing*, vol. 26, no. 5, p. 1484–1509, 1997.

[3] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[4] M. A. Nielsen and I. Chuang, "Quantum Computation and Quantum Information," 2002.

[5] E. G. Rieffel and W. H. Polak, *Quantum Computing: A Gentle Introduction*. MIT Press, 2011.

[6] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[7] A. W. Harrow, A. Hassidim, and S. Lloyd, "Quantum Algorithm for Linear Systems of Equations," *Physical review letters*, vol. 103, no. 15, p. 150502, 2009.

[8] R. LaRose, "Overview and Comparison of Gate Level Quantum Software Platforms," *Quantum*, vol. 3, p. 130, 2019.

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

[10] National Academies of Sciences, Engineering, and Medicine, *Quantum Computing: Progress and Prospects*. The National Academies Press, 2019.

[11] F. Leymann and J. Barzen, "The bitter truth about gate-based quantum algorithms in the nisq era," *Quantum Science and Technology*, vol. 5, no. 4, p. 044007, 2020.

[12] K. Mitarai, M. Kitagawa, and K. Fujii, "Quantum analog-digital conversion," *Physical Review A*, vol. 99, no. 1, p. 012301, 2019.

[13] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.

[14] C. A. Ellis, "Workflow Technology," *Computer Supported Cooperative Work, Trends in Software Series*, vol. 7, pp. 29–54, 1999.

[15] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 1999.

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

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

[18] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[19] E. Farhi, J. Goldstone, and S. Gutmann, "A Quantum Approximate Optimization Algorithm," *arXiv:1411.4028*, 2014.

[20] S. Endo, S. C. Benjamin, and Y. Li, "Practical Quantum Error Mitigation for Near-Future Applications," *Physical Review X*, vol. 8, no. 3, p. 031027, 2018.

[21] D. Aharonov, W. Van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev, "Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation," *SIAM review*, vol. 50, no. 4, pp. 755–787, 2008.

[22] H. J. Briegel, D. E. Browne, W. Dür, R. Raussendorf, and M. Van den Nest, "Measurement-based quantum computation," *Nature Physics*, vol. 5, no. 1, pp. 19–26, 2009.

[23] J. Gruska, *Quantum Computing*. Citeseer, 1999, vol. 2005.

[24] M. Mosca, "Quantum Algorithms," *arXiv:0808.0369*, 2008.

[25] F. Leymann, "Towards a Pattern Language for Quantum Algorithms," in *Quantum Technology and Optimization Problems*. Springer International Publishing, 2019, pp. 218–230.

[26] F. B. Maciejewski, Z. Zimborás, and M. Oszmaniec, "Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography," *Quantum*, vol. 4, p. 257, 2020.

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

[28] F. Leymann, D. Roller, and M.-T. Schmidt, "Web services and business process management," *IBM systems Journal*, vol. 41, no. 2, pp. 198–211, 2002.

[29] J. Eder and W. Liebhart, "Workflow Recovery," in *Proceedings First IFCIS International Conference on Cooperative Information Systems*. IEEE, 1996, pp. 124–134.

[30] K. Görlach, M. Sonntag, D. Karastoyanova, F. Leymann, and M. Reiter, "Conventional Workflow Technology for Scientific Simulation," in *Guide to e-Science*. Springer, 2011, pp. 323–352.

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

[32] H. Eberle, T. Unger, and F. Leymann, "Process Fragments," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2009, pp. 398–405.

[33] U. Breitenbücher, P. Hirmer, K. Képes, O. Kopp, F. Leymann, and M. Wieland, "A Situation-Aware Workflow Modelling Extension," in *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services (iiWAS)*, 2015, pp. 1–7.

[34] Camunda. (2020) BPMN Workflow Engine. [Online]. Available: https://camunda.com/products/bpmn-engine

[35] M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, and K. Wild, "The NISQ Analyzer: Automating the Selection of Quantum Computers for Quantum Algorithms," *CCIS Communications in Computer and Information Science*, 2020.

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

[37] B. Malaescu, "An iterative, dynamically stabilized method of data unfolding," *arXiv:0907.3791*, 2009.

[38] L. Harzenetter, U. Breitenbücher, M. Falkenthal, J. Guth, C. Krieger, and F. Leymann, "Pattern-based deployment models and their automatic execution," in *11th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. IEEE Computer Society, 2018, pp. 41–52.

[39] D. R. Simon, "On the Power of Quantum Computation," *SIAM journal on computing*, vol. 26, no. 5, pp. 1474–1483, 1997.

[40] E. Bernstein and U. Vazirani, "Quantum Complexity Theory," *SIAM Journal on computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[41] J. Du, M. Shi, X. Zhou, Y. Fan, B. Ye, R. Han, and J. Wu, "Implementation of a quantum algorithm to solve the Bernstein-Vazirani parity problem without entanglement on an ensemble quantum computer," *Physical Review A*, vol. 64, no. 4, p. 042306, 2001.

[42] G. Brassard, P. Hoyer, M. Mosca, and A. Tapp, "Quantum Amplitude Amplification and Estimation," *Contemporary Mathematics*, vol. 305, pp. 53–74, 2002.

[43] J. R. McClean, J. Romero, R. Babbush *et al.*, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, p. 023023, 2016.

[44] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. Gambetta, "Hardware-efficient Variational Quantum Eigensolver for Small Molecules and Quantum Magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[45] S. Mazanek and M. Hanus, "Constructing a Bidirectional Transformation between BPMN and BPEL with a Functional Logic Programming Language," *Journal of Visual Languages & Computing*, vol. 22, no. 1, pp. 66–89, 2011.

[46] C. Ouyang, M. Dumas, A. H. Ter Hofstede, and W. M. Van der Aalst, "From BPMN Process Models to BPEL Web Services," in *2006 IEEE International Conference on Web Services (ICWS)*. IEEE, 2006, pp. 285–292.

[47] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[48] V. Dunjko and H. J. Briegel, "Machine learning & artificial intelligence in the quantum domain: a review of recent progress," *Reports on Progress in Physics*, vol. 81, no. 7, p. 074001, 2018.

[49] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones *et al.*, "Scientific workflow management and the Kepler system," *Concurrency and computation: Practice and experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[50] Zapata. (2020) Orquestra. [Online]. Available: https://www.zapatacomputing.com/orquestra

[51] T. Häner, D. S. Steiger, K. Svore, and M. Troyer, "A software methodology for compiling quantum programs," *Quantum Science and Technology*, vol. 3, no. 2, p. 020501, 2018.

[52] G. Falazi, M. Hahn, U. Breitenbücher, and F. Leymann, "Modeling and execution of blockchain-aware business processes," *SICS Software-Intensive Cyber-Physical Systems*, vol. 34, no. 2-3, pp. 105–116, 2019.

[53] R. J. Sethi, H. Jo, and Y. Gil, "Re-Using Workflow Fragments Across Multiple Data Domains," in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE, 2012, pp. 90–99.

[54] A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik, "Dynamic Adaptation of Fragment-based and Context-aware Business Processes," in *2012 IEEE 19th International Conference on Web Services*. IEEE, 2012, pp. 33–41.

[55] K. Képes, U. Breitenbücher, S. G. Sáez, J. Guth, F. Leymann, and M. Wieland, "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, pp. 69–83.

[56] D. Schumm, D. Karastoyanova, F. Leymann, and S. Strauch, "Fragmento: Advanced Process Fragment Library," in *Proceedings of the 19th International Conference on Information Systems Development*. Springer, 2010.

[57] D. Garijo, O. Corcho, and Y. Gil, "Detecting common scientific workflow fragments using templates and execution provenance," in *Proceedings of the seventh international conference on Knowledge capture*, 2013, pp. 33–40.

[58] J. Wen, Z. Zhou, F. Lei, and J. Zhang, "Basic and personalized pattern-based workflow fragments discovery," *Personal and Ubiquitous Computing*, pp. 1–21, 2019.

All links were last followed on October 1, 2020.