Institute of Architecture of Application Systems

Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective

Benjamin Weder, Johanna Barzen, Frank Leymann, Michael Zimmermann

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

BibTeX:

```
@inproceedings{Weder2021_OrchestrationsInSuperposition,
  Author    = {Weder, Benjamin and Barzen, Johanna and Leymann, Frank and
               Zimmermann, Michael},
  Title     = {{Hybrid Quantum Applications Need Two Orchestrations in
               Superposition: A Software Architecture Perspective}},
  Booktitle = {Proceedings of the 18\textsuperscript IEEE International
               Conference on Web Services (ICWS 2021)},
  Year      = {2021},
  Pages     = {1--13},
  Doi       = {10.1109/ICWS53863.2021.00015},
  Publisher = {IEEE}
}
```


Universität Stuttgart
Germany

Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective

Benjamin Weder, Johanna Barzen, Frank Leymann, and Michael Zimmermann
University of Stuttgart, Institute of Architecture of Application Systems,
Universitätsstraße 38, Stuttgart, Germany
{*weder, barzen, leymann, zimmermann*}*@iaas.uni-stuttgart.de*

***Abstract*—Quantum applications are most often hybrid, i.e., they are not only made of implementations of pure quantum algorithms but also of classical programs as well as workflows and topologies as key artifacts, and data they process. Since workflows and topologies are referred to as "orchestrations" in modern terminology (but with very different meanings), two orchestrations that go hand-in-hand are required to realize quantum applications. We motivate this by means of a non-trivial example, sketch these orchestration technologies, and reveal the overall structure of non-trivial quantum applications. Furthermore, we discuss the implied architecture of a runtime environment for such quantum applications. To validate the introduced architecture, we present a prototypical implementation based on the Camunda workflow engine, its associated modeling tool, as well as the OpenTOSCA ecosystem.**

***Keywords*-Quantum Computing, NISQ, Software Engineering of Quantum Applications, Hybrid Quantum-Classical Applications, Runtime for Quantum Applications**

I. Introduction

Nowadays, quantum applications are in most cases *hybrid*, i.e., they encompass not only implementations of one or more quantum algorithms but require classical programs as well in order to produce their final result [1], [2], [3]. This is most evident by the requirement for pre- and post-processing. For example, pre-processing generates quantum circuits for *state preparation* within a classical environment and prepends these quantum circuits to the quantum algorithm proper [1], [4], [5]. These state preparation circuits then create - when being executed - the quantum state that represents the input to be processed by the quantum algorithm in the register of the quantum computer [3], [6]. An example for post-processing is the correction of readout-errors within a classical environment by applying an *unfolding technique* to compute the less disturbed result distribution from the disturbed measured distribution [7], [8].

But even algorithms that are often considered as "proper quantum" algorithms are in fact hybrid. For example, the factorization algorithm of *Shor* [9] consists of a quantum part utilizing a *Quantum Fourier Transform (QFT)* [10] to find the period of a function. After the quantum computation, the output must be post-processed by a classical program by means of analyzing continued fractions [9], [11]. Therefore, the implementation of this quantum algorithm needs to integrate with a classical program to produce its final result.

In general, a hybrid quantum application (or *quantum application* for short) is not only made of implementations of quantum algorithms (called *quantum programs* from here on) and classical programs. Furthermore, it also comprises data to be processed, workflows, and topology models. The quantum programs may be written in a quantum assembler, such as *OpenQASM*[1] or *Quil*[2], or a host programming language using quantum libraries, such as *Qiskit*[3] or *Cirq*[4] [12], [13], [14]. Additionally, the classical programs can be implemented in any programming language and run in any execution environment. The data to be processed may be provided by value or by reference (and then retrieved), have to be properly transformed, etc. Workflows may be used for preparing data for further processing, for controlling the execution order of the quantum programs and classical programs, as well as passing data between these programs [15]. Finally, topology models enable to provision the execution environment for the quantum and classical programs [16].

In this paper, we extend [17] and (i) present a non-trivial quantum application from the domain of the *humanities* [18] that motivates the use of *workflow technology* [19], [20] as a major enabler for real-world quantum applications. Further, (ii) the need for *provisioning technology* as the other major enabler is described. Thereby, it enables the automatic provisioning of the required execution environments for quantum and classical programs, as well as their update, e.g., if a new software version is available [21], [22]. Moreover, we (iii) discuss the implied architecture of a modeling and runtime environment for such quantum applications. Finally, (iv) a prototypical implementation of the introduced architecture and our sample quantum application are presented.

This paper is structured as follows: Section II sketches fundamentals about workflow technology and introduces our sample quantum application. Then, Section III describes the need for provisioning technology to set up the environment to execute a workflow. In Section IV, the runtime environment for quantum applications is detailed, and Section V presents our prototypical implementation. Section VI discusses related work and we conclude in Section VII.

[1]https://github.com/qiskit/openqasm
[2]https://github.com/rigetti/quil
[3]https://qiskit.org
[4]https://github.com/quantumlib/cirq

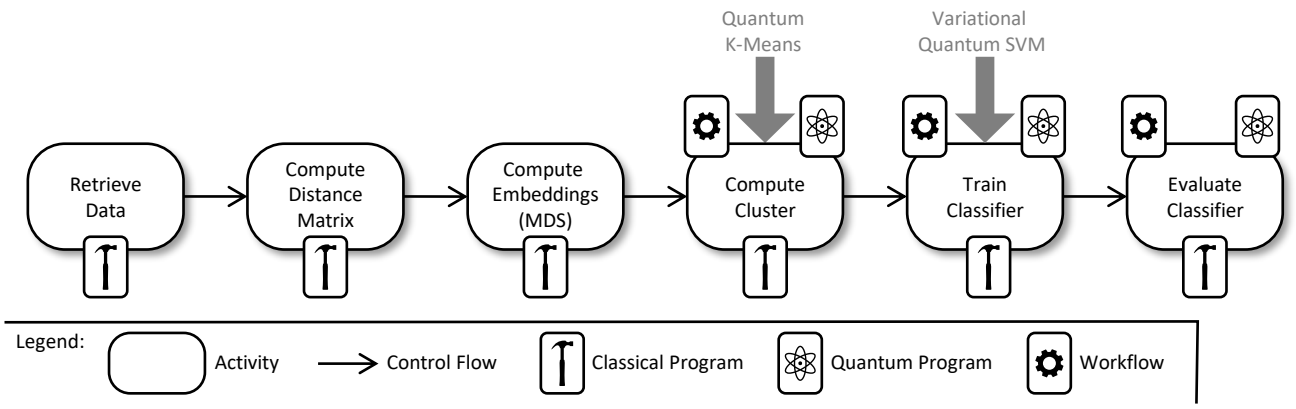

Figure 1. The Workflow of the Sample Hybrid Quantum Application.

II. Workflows: Orchestrating Control- and Data Flow

In this section, we briefly sketch the concept of a workflow model and a workflow instance. Furthermore, we describe a real-world sample hybrid quantum application.

A. Workflows in a Nutshell

Workflow technology is well-established since decades [19], [20]. In a nutshell, it is a technology to specify the partial order of a collection of *activities* that have to be performed to achieve a composite goal. The partial order is based on *control flow* dependencies between the activities [19]. Typically, the activities are represented as nodes in a directed graph (see Figure 1), and the control flow dependencies are the edges of the graph [23]. Such an edge points from a certain activity to those activities that may have to be performed once the source activity finished successfully [20]. Whether or not a target activity is performed is controlled by a Boolean condition associated with the corresponding edges [19]. This condition is evaluated based on data that has been returned by already finished activities. This way, the set of activities performed by a workflow is highly dependent on the results of the activities. Consequently, the actual paths taken through the graph typically change from execution to execution of the workflow [20]. The directed graph representing the workflow is referred to as a *workflow model*, and an execution of such a workflow model is referred to as an *instance* of the model [19]. Nowadays, workflow models are usually specified in *BPMN* [24], which is a graphical language with an operational semantics describing how instances of a workflow graph are created. [23] gives an overview of several key languages for specifying workflow models. Thereby, workflow technology provides several benefits, such as robustness, reliability, and scalability [19], [25]. Furthermore, it allows the specification of alternative control flows in the case of an error [20], the definition of transactions comprising multiple activities [26], as well as compensation actions for different activities [19], [27].

B. A Sample Hybrid Quantum Application

Figure 1 shows a sample hybrid quantum application applying quantum machine learning in the domain of the humanities [18]. The presented quantum application will perform clustering on a set of input data, and based on the clustering results, train a classifier for the classification of further data. As input data, a subset from the *MUSE*[5] repository [28] is used, which contains data about costumes identified by analyzing various films [29]. Therefore, the first activity of the workflow retrieves the input data from the MUSE repository. This activity is implemented by a classical program, as indicated by the ”hammer” icon associated with the activity. The retrieved costume data are often categorical, e.g., the different colors of pieces of clothes comprising a costume [12]. However, most of the (quantum) machine learning algorithms require numerical or metrical data [18]. Thus, the categorical data must be transformed into numerical data before applying the machine learning algorithms [30]. For this, the second activity of the workflow computes the distance matrix of this categorical data based on the *Wu-Palmer similarity measure* [31]. Then, *multidimensional scaling (MDS)* [32] is applied to reduce the dimension of the feature space, which is implemented by a classical program too. After that, clusters in the data are determined, which involves a sub-workflow (indicated by the “gear” icon) utilizing the *negative rotation quantum k-means algorithm* [33]. This sub-workflow comprises activities that are implemented by classical programs, as well as quantum programs (indicated by the ”atom” icon), and is discussed in more detail in Section V-B. The results of the clustering sub-workflow are used by another sub-workflow, which trains a classifier using a *variational quantum support vector machine (SVM)* [34]. Finally, the trained classifier is evaluated. This includes, e.g., classical programs generating test data, and quantum programs performing the classification with the trained parameterization of the variational quantum SVM.

[5]https://www.iaas.uni-stuttgart.de/en/projects/muse

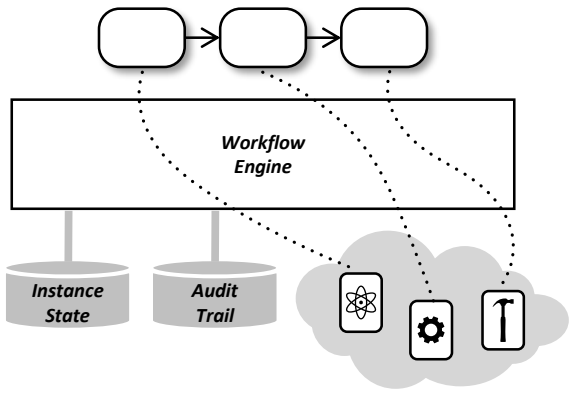

Figure 2. Executing a Workflow.

C. Executing a Workflow

A workflow model will be executed by a *workflow engine* [19], [20], [25]. For this purpose, the workflow engine navigates through the workflow model: For example, it determines the activities that are ready to be performed, collects their input data, starts (and controls) their execution (in parallel), and retrieves their output. Further, the workflow engine detects once an activity completes successfully - based on the transition conditions of the edges leaving a completed activity - the activities to be executed next. Such a workflow in execution is referred to as an instance of the corresponding workflow model (see [19] for the details). Note that especially in a cloud environment workflows are also called *orchestrations* - based on the mental model that a workflow "orchestrates" all the actions required to create a single whole from the executions of the individual activities.

Obviously, the workflow engine must know how an activity is implemented. Therefore, the workflow model associates with each activity its implementation, e.g., a classical program or a quantum program, or it specifies how an implementation can be discovered at runtime, e.g., using a *service registry* [35], [36]. Once an activity is determined to be ready for execution, its input data is gathered by the workflow engine, transformed into a format required by its implementation, and passed to the implementation [19], [37]. This implies that the workflow engine understands the invocation mechanism of the implementation [20], [38]. For example, how to call a Java or Python program, how to serve a REST API, how to start another workflow, how to communicate via a message queue, and so on. The different implementations of the various activities may not only be very heterogeneous but especially highly distributed, and they may run in very different environments. Furthermore, some of the activity implementations may be long-running and return responses asynchronously at unforeseen times, which implies that the workflow engine must understand how to correlate incoming data with workflow instances and running activities therein to detect their completion [19].

This in turn requires that the state of a workflow instance (consisting of the state of each of its activities, the input and output data of the activities, and so on) must be made persistent by the workflow engine [19]. Especially, this implies that the execution of a workflow is interruptible. Also, errors are detected by the workflow engine, and parts of a workflow instance can be automatically undone if such an error is detected [20]. This requires that for activities to be undone a compensation activity is assigned, which the workflow engine will invoke in case of an error [27]. Activities may be grouped into corresponding units of work that have transactional semantics. Thus, a workflow is recoverable.

Figure 2 shows a workflow engine and a workflow model navigated by this engine. The implementations corresponding to the activities are available to the workflow engine. When navigating the workflow model, the state information about the corresponding instance is stored in a database (see *instance state*). This allows to monitor running workflow instances [19]. Once an instance is completed, the information about the history of its execution (i.e., the steps performed, their duration, input/output data, reasons for taking a particular path, etc.) is moved to another database referred to as *audit trail* [39]. The audit trail can be analyzed to improve workflows (making it faster, cheaper, correcting modeling errors, etc.) and enable reproducibility of results [40], [41].

III. Provisioning: Orchestrating Topology Deployment

In this section, we sketch how the environment required for executing a workflow is specified and automatically set up. Furthermore, we discuss the structure of an archive to package hybrid quantum applications with all dependencies.

A. Topology of a Hybrid Quantum Application

Whenever an activity of a workflow is detected to be ready for execution, the corresponding activity implementation is invoked by the workflow engine (see Section II-C). This assumes that the corresponding implementation is available in (or at least accessible from) the environment [19]. Since an activity implementation (e.g., a Java program) typically has dependencies on other components (like a JVM), these components have to be available too - and in a transitive manner: Only if all these components are present and intertwined correctly, the activity implementation can be performed [20]. All the necessary components and their dependencies are described by a directed acyclic graph (DAG) [42]. Thereby, the nodes are the components, and the directed edges are the dependencies between them. Such a graph is referred to as a *topology model* or *declarative deployment model* respectively [16], [43]. To define such topology models and automatically provision the described application, a plethora of *provisioning* or *deployment technologies*, such as Kubernetes, Puppet, Terraform, or Ansible, have been proposed and are used in practice today [21], [44].

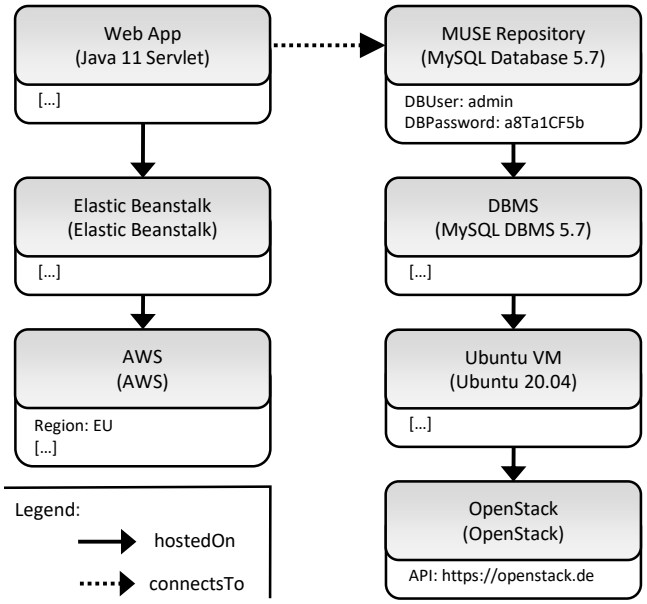

Figure 3. Exemplary Topology Model.

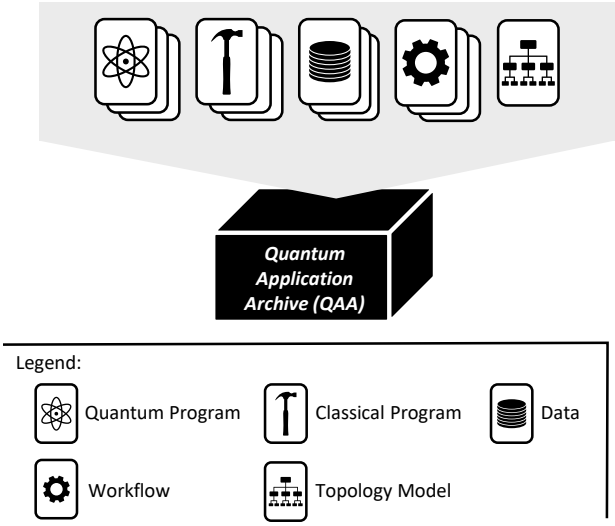

Figure 4. Structure of a Quantum Application Archive (QAA).

Additionally, vendor- and technology-agnostic standards to specify topology models, such as the *Topology and Orchestration Specification for Cloud Applications (TOSCA)* [45], [46], have been developed to ensure portability and interoperability between different environments [16], [22]. Using TOSCA, the topology model of an activity implementation, or application in general, is defined using a so-called *topology template* [45]. Thereby, the nodes in a topology template are called *node templates* and the edges *relationship templates*. The semantics of node templates, as well as relationship templates, are defined by reusable *node types* and *relationship types*. Figure 3 depicts an example of a topology template using the graphical notation introduced in [47], whereby the types of the node templates are shown in braces while types of the relationships are encoded by their line type. The topology template shows a possible implementation of the ”Retrieve Data” activity of the sample workflow from Figure 1 and its dependencies. The activity is implemented by a Java web app, which shall be hosted on Elastic Beanstalk, a Platform as a Service (PaaS) offering from AWS [48]. Furthermore, the Java web app needs to connect to the MUSE repository to retrieve the required data (see Section II-B). The database is managed by a MySQL database management system, that will be hosted on an Ubuntu virtual machine, which in turn is created using OpenStack. To enable the configuration of node and relationship templates, their corresponding types expose so-called *properties* that allow setting configuration parameters. For example, the MUSE repository node template in Figure 3 is configured with a username and password that can be used to access the database after the provisioning.

B. Package of a Hybrid Quantum Application

Hybrid quantum applications are delivered as a single entity referred to as a *quantum application archive (QAA)*. This archive is a self-contained package that encompasses all the artifacts needed to set up the execution environment required to perform the quantum application. The conceptual structure of a QAA is depicted in Figure 4. First, this package contains the topology model describing the components and their dependencies. Next, all quantum programs, as well as all classical programs making up the quantum application, are included (or pointed to). Additionally, the workflow model orchestrating the execution of the activity implementations are in the package, as well as workflow models that are (re-)used as sub-workflows (like the ”Train Classifier” sub-workflow realizing a variational quantum SVM in Figure 1). Finally, some quantum applications like a machine learning application for training a neural net may need special data, and such data can also be included in the package. This way a quantum application becomes an entity like an app that can be stored somewhere, e.g., in some sort of a quantum app store, advertised, bought, and so on [6].

C. Provisioning an Execution Environment

Before the quantum application corresponding to a quantum application archive can be executed, its execution environment must be set up [49]. The manual provisioning of all required components is complex, time-consuming, and error-prone [42]. Therefore, to automate this process, the topology model of the execution environment is interpreted by a *provisioning engine* [21]. In a nutshell, the provisioning engine interprets the topology model ”from the bottom to the top”, i.e., starting from the leaves of the topology graph in the reverse direction of the edges. For each node visited, the corresponding component or artifact is installed [44].

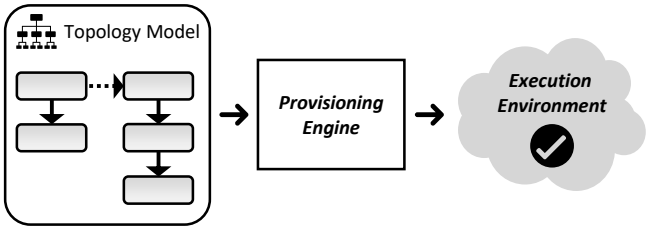

Figure 5. Provisioning of a Quantum Application Archive.

For example, in Figure 3, the code of the Java web app is uploaded to AWS Elastic Beanstalk and the application provisioning is triggered. In parallel, the virtual machine for the database is created in the specified OpenStack. Then, the database management system is installed, the database is set up, and it is initialized with the data from a backup. Finally, the connection between the Java web app and the database is established such that it can access the database and retrieve data at runtime. Figure 5 depicts that the provisioning engine interprets a topology model and provisions the corresponding execution environment. Note that in analogy to workflows, the interpretation of topology models for provisioning an execution environment is called *orchestration* too - the mental model is again that the provisioning engine "orchestrates" all the actions required to create an execution environment as a whole by deploying individual artifacts. See [22], [42], [44] for more details on the automated provisioning of applications, services, or execution environments using a provisioning engine.

D. Managing an Execution Environment

In addition to the initial provisioning of an execution environment, it has to be managed afterwards, which can also be automated using a provisioning engine [22], [50], [51]. For example, the database in Figure 3 may require periodic backups, which can be annotated in the corresponding node of the topology model [50], [52]. Therefore, the provisioning engine periodically retrieves the database state and persists it at a defined location. Further, if a new version of the used database management system is released, the node type in the topology model can be changed to the new version. Then, it is automatically updated by the provisioning engine while ensuring that the state in the database is preserved [52]. Such updates are especially important in the quantum computing domain, where new versions of software tools or libraries, such as Qiskit, are released frequently [14]. Hence, the application developer is not in charge of performing a manual update. Another management task accomplished by the provisioning engine may be the scaling of some components of the topology model, e.g. if the workload gets too high [53]. Finally, the provisioning engine can also be used to decommission an execution environment after the quantum application has terminated to save resources [22].

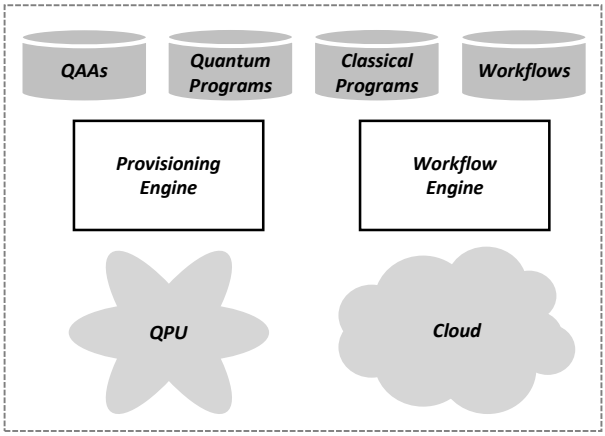

Figure 6. Overview of a Hybrid Quantum-Classical Runtime Environment.

IV. The Hybrid Quantum-Classical Environment

The technologies and concepts that have been described in the previous sections of this paper imply architectural components of an environment for executing hybrid quantum applications. In this section, we outline these implications.

A. High-Level Architecture

The high-level architecture of a runtime environment for hybrid quantum applications is depicted in Figure 6. Every quantum application includes quantum programs, thus, the runtime environment has to encompass one or more *quantum processing units (QPUs)*. Since QPUs are nowadays made available by means of cloud access [6], [54], it is only natural to assume that the classical programs of a quantum application are running in a cloud environment too. Note that the latter is without loss of generality because workflow engines can also invoke implementations that run in non-cloud environments [19]. Having said that, the runtime environment obviously must contain a workflow engine. To set up the execution environment for running the implementations of the activities of the workflow, a provisioning engine has to be available. Both, the workflow engine as well as the provisioning engine are also hosted in the cloud.

Consequently, all the artifacts that make up a quantum application have to be accessible in the cloud. First of all, the QAAs are needed by the provisioning engine to set up the execution environment for the activity implementations of the workflows of the quantum applications. The provisioning engine will process the QAA of a quantum application by orchestrating the deployment of the included topology. During this processing, the quantum programs and classical programs will be installed, and their prerequisites will be made available (see Section III-C). Next, the workflow engine will instantiate the workflow model representing the quantum application, i.e., the workflow models have to be part of the overall runtime environment too.

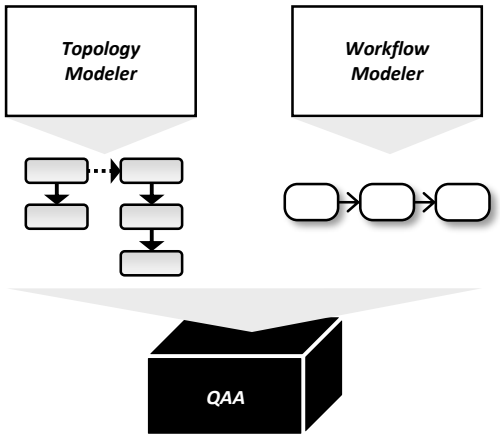

Figure 7. Modeling Environment for Hybrid Quantum Applications.

In addition to the runtime environment, an environment for specifying the artifacts comprising a quantum application is needed. Besides tools well-known to quantum programmers like a circuit designer, modeling tools for topologies and workflows are required. Figure 7 shows these components of the modeling environment and that the resulting topologies and workflows can be packaged into a QAA.

B. Running a Hybrid Quantum Application

Figure 8 depicts how the execution of a hybrid quantum application is kicked off. A corresponding *RUN* message defining the name of the workflow Ω and the initial parameter values $p_1, \ldots, p_k$ to be passed to the newly created workflow instance is put into a queue (see ①). This queue is the entry into the hybrid quantum-classical environment.

A dedicated component, which we refer to as the *queue controller*, monitors the queue, analyzes the messages, and forwards them to the responsible components for further processing. In our context, the queue controller understands that the message solicits to instantiate the workflow model Ω and, therefore, passes a corresponding request to the workflow engine (see ②). The workflow engine will fetch the workflow model Ω and will create a new instance passing the parameter values $p_1, \ldots, p_k$ to it as input data (see ③).

This assumes that the environment needed by the workflow for its execution has already been provisioned. To reduce the monetary costs incurred by cloud resources for workflows that are only rarely performed, the environment may be created for each execution of a workflow and can be decommissioned once the workflow finishes (see Section III-D). If a workflow model is instantiated and executed very often, decommissioning the corresponding environment and provisioning it over and over again may turn out to be a significant overhead and should be avoided. However, a component like a *resource manager* may be in charge of the decision when to decommission an environment [55].

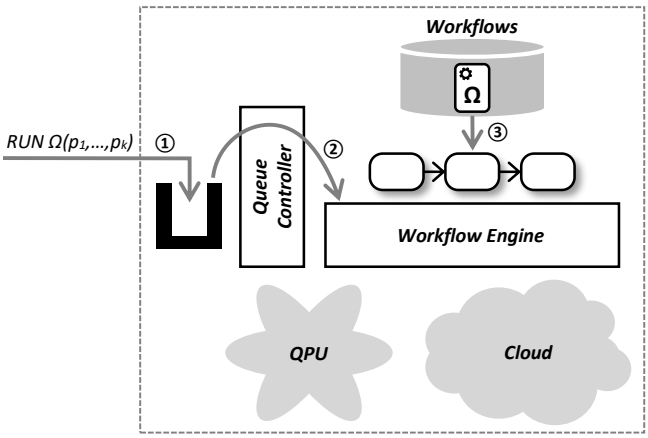

Figure 8. Starting the Execution of a Hybrid Quantum Application.

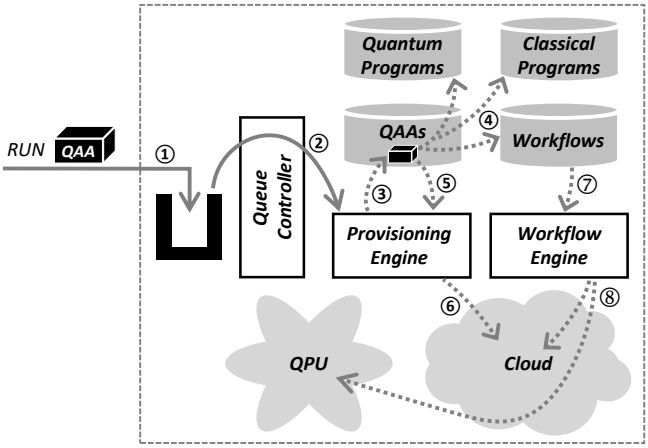

Figure 9. Passing an Application Archive for Immediate Execution.

The above requires that the quantum application archive has been unpacked before and that its encompassed artifacts are accessible to the provisioning engine, e.g., stored in the environment. If this is not intended (e.g., in order to avoid storage costs), another possibility is a variant of the RUN message that allows putting a complete quantum application archive for processing into the queue (see ① in Figure 9).

In this case, the queue controller will request the provisioning engine to deploy the quantum application archive (see ②) and next request the workflow engine to run the contained hybrid quantum application. From outside, this is perceived as a single step [56]: The QAA will be stored (temporarily) to be able to retry deploying the QAA in case of errors (see ③). Next, the archive is unpacked (see ④): The quantum programs, the classical programs, and the workflows are stored such that they are individually accessible. In step ⑤, the provisioning engine will interpret the topology model of the QAA and determine all actions needed to set up the execution environment required by the hybrid quantum application. Once

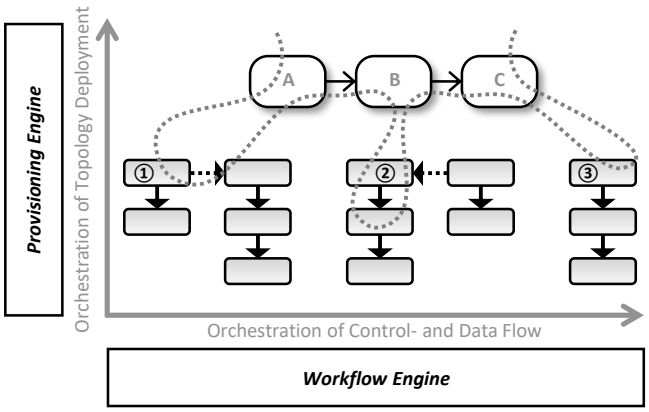

Figure 10. Executing a Hybrid Quantum Application Requires Two Orchestrations in Superposition.

all these actions are performed, the environment needed by the workflow for its execution is deployed (see ⑥). Afterwards, the main workflow model of the hybrid quantum application will be instantiated (see ⑦) and executed. During the workflow execution, the workflow engine will invoke classical programs deployed in the cloud and will kick off the execution of quantum programs on a QPU (see ⑧).

C. Hybrid Quantum Applications and Two Orchestrations in Superposition

As a consequence, two different kinds of orchestrations are required to perform a hybrid quantum application as depicted in Figure 10: (i) The orchestration of the control- and data flow between the activities of the quantum application, and (ii) another orchestration of the topology provisioning for the execution environment required by the workflow to be executed. The figure also shows that these two kinds of orchestrations are intertwined [38], [57], they are in a loose sense in superposition: The orchestration performed by the workflow engine (the x-axes) goes hand in hand with the orchestration performed by the provisioning engine (the y-axes). The workflow engine instantiates the workflow model and invokes activity A, which requires that implementation ① of activity A is properly set up. This setup is based on the corresponding fragment of the topology of the overall environment needed by the workflow as indicated in the figure. Once activity A completed successfully, the implementation ② of activity B is started, which assumes a proper setup corresponding to another fragment of the overall topology. Finally, activity C's implementation ③ is kicked off, and the required setup of this implementation is specified by yet another topology fragment.

D. Optimization Potentials: Example

The graph-based nature and corresponding operational semantics of (many) orchestration languages support their formal analysis for predictions or improvements, for example. This offers opportunities for improved support of quantum programs that iteratively access a quantum computer, such as implementations of *variational quantum algorithms* that are often used today with NISQ devices [2], [58], [59]. The iterative nature of such an algorithm, i.e., the invocation of quantum programs in a loop that typically also contains classical programs, can be detected in a workflow model.

In such a situation, the hybrid quantum-classical environment may reserve sole access to the quantum computer for the corresponding quantum application and provide direct access to the quantum computer without having to submit requests via the queue (see Section IV-B). This will reduce the latency of the hybrid quantum application significantly.

V. Prototypical Validation

In this section, we first introduce the different components prototypically realizing the presented architecture of a hybrid quantum-classical modeling and runtime environment. Afterwards, we present the prototypical implementation of the sample hybrid quantum application depicted in Figure 1.

A. Realizing the Hybrid Quantum-Classical Environment

To verify the sketched architecture of an environment to model and execute hybrid quantum applications, we introduce our corresponding prototype in what follows. For modeling the orchestrations of the control- and data flow of a quantum application, the workflow language BPMN [24] has been chosen. Thus, the Camunda BPMN workflow engine [60] and its associated modeling tool [61] can be used for defining and running the workflows of a hybrid quantum application. The topologies of the quantum and classical programs are specified using the TOSCA standard [45]. For this, we utilize the open-source TOSCA modeling tool *Winery*[6] [62], which is part of the *OpenTOSCA ecosystem* [63]. The OpenTOSCA ecosystem also comprises the *OpenTOSCA Container*[7] [64], a TOSCA-compliant provisioning engine, which is used to provision and manage the execution environments of the hybrid quantum applications.

After modeling the workflows and topologies, all required artifacts for the hybrid quantum application must be packaged in the QAA. Thus, the workflow models are uploaded to Winery, from where all artifacts can be exported as a self-contained *Cloud Service Archive (CSAR)*, the packaging format defined by the TOSCA standard [45], [65]. This means QAAs are an extension of CSARs, providing a self-contained packaging format for quantum applications. The CSARs can then be uploaded to the OpenTOSCA Container to provision the execution environment, deploy the workflows to the Camunda workflow engine, and perform the binding between the workflows and the execution environment. Thereby, the integration of the queue controller into this toolchain is part of our future work (see Section IV-B).

[6]https://github.com/OpenTOSCA/winery
[7]https://github.com/OpenTOSCA/container

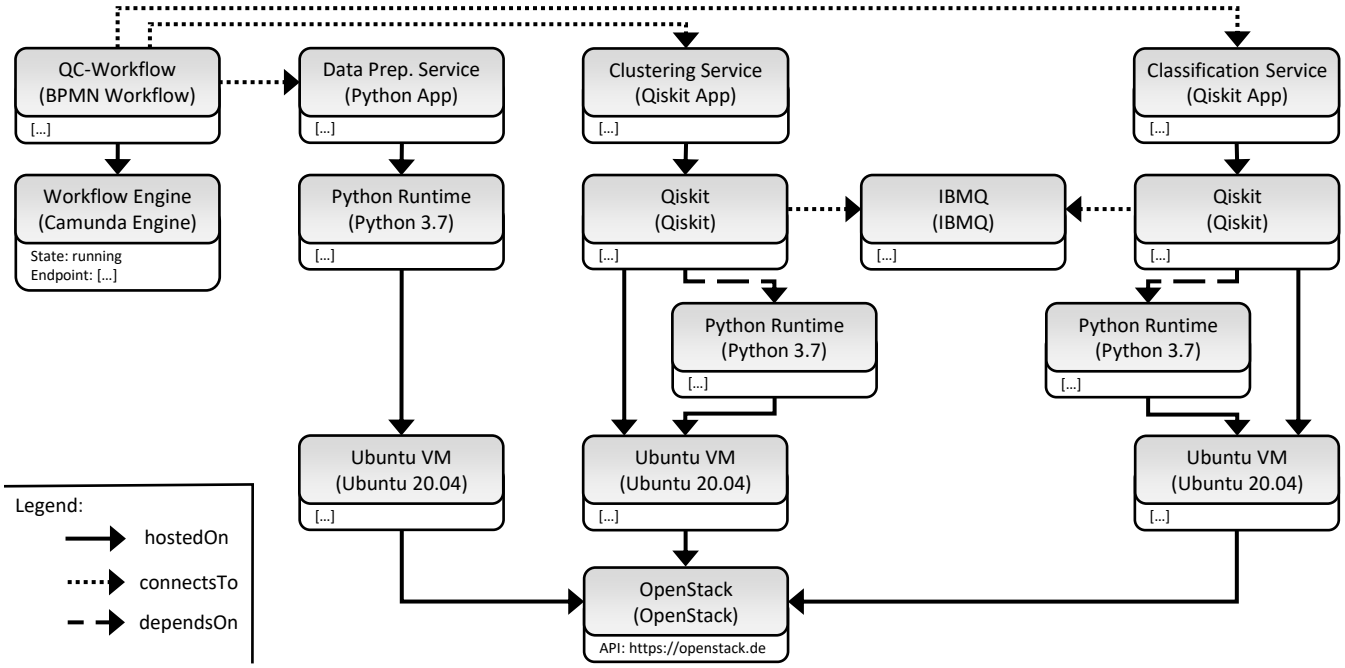

Figure 11. The Topology Model of the Sample Hybrid Quantum Application.

B. Implementation of the Sample Quantum Application

In the following, we discuss implementation details of the sample hybrid quantum application introduced in Section II-B and for which the corresponding workflow is depicted in Figure 1. Due to space reasons, only some aspects can be considered, especially just one of the three sub-workflows contained in the overall workflow. However, the complete workflow model, the corresponding topology model, and detailed instructions on how to execute the hybrid quantum application are available on Github[8].

1) Topology Model of the Sample Quantum Application: Figure 11 shows the topology model of the sample hybrid quantum application. The first topology stack on the left comprises the workflow engine used for the execution of the hybrid quantum application and the corresponding workflow model. Thus, this stack instructs the provisioning engine to upload the workflow model to the workflow engine and to perform the binding between the workflow and the activity implementations. Thereby, the state of the workflow engine is set to "running", indicating that the workflow engine is already available in the hybrid quantum-classical environment and does not have to be deployed by the provisioning engine. However, it also enables to provision a workflow engine on-demand by removing this property and adding a corresponding hosting environment, e.g., a virtual machine. Finally, the stack can also be removed from the topology model if the queue controller is available and performs the upload of the workflow model to the workflow engine.

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

The second topology stack contains a *data preparation service* wrapping the required logic for the first three activities of the sample workflow (see Figure 1). This service is implemented as a Python application, and thus, hosted on a corresponding Python runtime. The Python runtime and the application are provisioned on an Ubuntu virtual machine, which is created using OpenStack. Furthermore, the connection between the workflow and the data preparation service specifies that the workflow uses the service at runtime and is configured correspondingly by the provisioning engine [66]. Note that we use a subset of the data from the MUSE repository that is directly accessible on Github[8] and *DaRUS* [28] for our prototype. Thus, the data preparation service can load the data from there, and the database does not have to be provisioned every time the application is executed. Hence, the provisioning is more efficient for our sample quantum application and requires fewer virtual machines. When running the quantum application in a production environment, e.g., to classify customer data, the corresponding database can be added to the topology model as an additional topology stack. However, in such a scenario, the database is usually already running, and hence, the provisioning engine only configures the service to access the database at runtime, similar to the workflow in the first topology stack.

Additionally, the topology model comprises two stacks defining a *clustering service* implementing the different activities of the "Compute Cluster" sub-workflow and a *classification service* providing the functionality of the two remaining sub-workflows (see Figure 1). Both services are implemented using Qiskit and have a corresponding de-

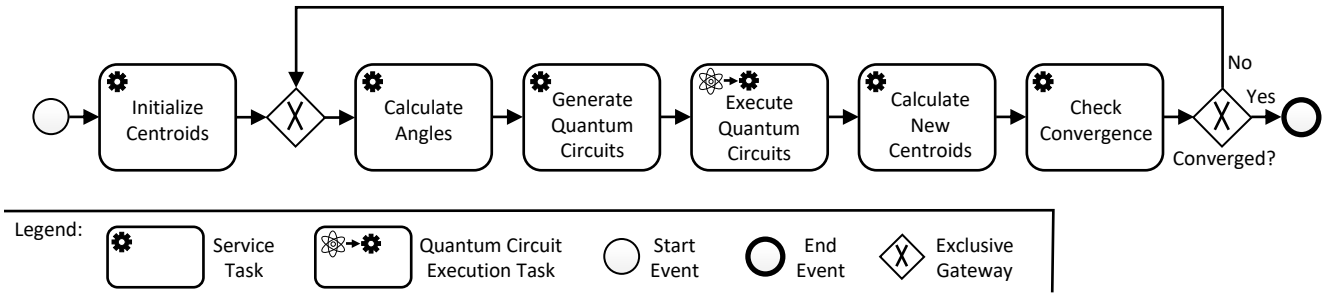

Figure 12. The Sub-Workflow of the "Compute Cluster" Activity Using the BPMN Concepts and their Graphical Notation.

pendency in their topology stack. The Qiskit node has a connection to an IBMQ node, indicating that it uses QPUs available over IBMQ for the execution of quantum circuits. Thereby, the IBMQ node enables to configure the quantum circuit execution, e.g., by defining the QPU to use or the access token in the properties [16]. However, otherwise, this can also be defined using input parameters of the workflow. Furthermore, Qiskit requires a Python runtime, which is installed on an Ubuntu virtual machine. For the sake of simplicity, all virtual machines are created in the same OpenStack, but the different services could also be hosted using other cloud offerings, e.g., AWS, as shown in Figure 3.

2) Implementation of the Clustering Sub-Workflow: Figure 12 depicts the sub-workflow implementing the "Compute Cluster" activity of the overall workflow. Thereby, the BPMN [24] concepts and their graphical notation are used. The sub-workflow can, e.g., be deployed as an independent workflow model and then invoked by a BPMN call activity from the overall workflow, or it may also be included as a BPMN sub-process. For the clustering, the negative rotation quantum k-means algorithm is implemented by the workflow (see [33] for details). First, the initial centroids for the k-means algorithm are determined by a *service task*, i.e., utilizing a classical program. Then, the workflow performs multiple iterations of quantum and classical processing until the result converges. Thereby, the angles on the unit sphere are calculated for the current centroids and the data points to cluster. Based on the resulting angles, quantum circuits are generated using another service task. For this, a corresponding *circuit template* is initialized with the angles as parameters [67], [68]. Next, the circuits are executed utilizing a so-called *quantum circuit execution task*, which is part of a BPMN extension for quantum computing (see [15], [49]). The results of the quantum circuit executions can be used to assign the data points to one of the clusters and to calculate new centroids depending on the assignments. Finally, the old and the new centroids are compared, and if the difference is larger than a given threshold, the loop is executed again, as indicated by the exclusive gateway in the workflow. When the algorithm converges, the final assignments of the data points to the clusters are returned.

VI. Related Work

A workflow language can be perceived as a parallel, long-running, interruptible, persistent, and recoverable programming language (see Section II-A). The instantiation of a workflow model by a workflow engine ensures these properties: The workflow engine acts like a virtual machine properly interpreting the workflow language, just like a Java Virtual Machine interprets the Java language. Attempting to use a traditional programming language to realize "workflows" will thus fail. Consequently, domains in which workflows play a central role either use an existing workflow system (i.e., a workflow engine and a matching modeling tool), or develop a separate workflow system that is targeted to the particular application domain. Note, that the latter is a huge endeavor. The use of an existing workflow system has the advantage that it is mature, proven, robust, etc. [19].

The latter situation occurred, e.g., in the domain of *e-Science* [69], [70] where a plethora of workflow systems (called *scientific workflow systems*) has been developed, such as *Taverna* [71], *Pegasus* [72], or *Kepler* [73]. As a result, workflows are hard to reuse across different domains of e-Science or various scientific workflow systems because these systems and their languages used are not standardized. Further, these workflow systems typically are not abreast in terms of maturity, etc., with conventional workflow systems. And it turned out that conventional workflow systems can be used for modeling and executing scientific workflows either without any modifications or with a few extensions [74].

In the quantum computing domain, history seems to repeat. First, workflow systems dedicated to quantum computing appeared in the scientific domain like *Nexus* [75]. But also product offerings specialized for quantum computing like Zapata *Orquestra* [76] are made available, providing a proprietary YAML-based language to define quantum workflows. In this context, too, it turned out that conventional workflow systems can be used for quantum workflows with only a very few extensions guaranteeing to benefit from the maturity and robustness of conventional workflow systems [15], [49]. Furthermore, by using standardized workflow languages like BPMN [24], the reuse of workflow models across different workflow engines is simplified.

Note, that nowadays conventional workflow systems support one of two standardized workflow languages: BPEL [77] or BPMN [24], while BPMN is becoming the dominant language. An overview of related standards and the concept behind the languages can be found in [23]. Using one of these two languages eases the reuse of workflows across supporting workflow engines. Extensions of BPMN for quantum computing have been proposed in [15], [49] and have been prototypically implemented based on the open-source BPMN workflow system Camunda [60], [61].

[57] introduced the concept of self-contained application archives, especially for the purpose of understandability and reproducibility of scientific in-silico experiments. This concept has been realized in [38] introducing self-contained archives consisting of workflows and all of their dependencies, both, in terms of modeling as well as automatically provisioning the complete environment required by a workflow for its execution. [43] and [78] propose a similar approach to define scientific workflows in a self-contained manner together with their corresponding execution environment. For this, they utilize the TOSCA standard to specify the topology for the workflow and directly model the control- and data flow within this topology by connecting the corresponding nodes. The resulting topology model is then interpreted by the TOSCA-compliant *Cloudify*[9] provisioning engine, which provisions the execution environment and enacts the workflow. However, the combination of both orchestrations in the topology model clutters the model and decreases its understandability. Furthermore, as already discussed, the features of mature workflow engines, e.g., their robustness, have then to be implemented in the provisioning engine.

In addition to the provisioning and deployment technologies presented in Section III-A, new approaches for the automated provisioning of quantum applications or corresponding extensions for existing technologies were developed. [16] presents *TOSCA4QC*, introducing two modeling styles to define hybrid quantum applications using TOSCA and to automatically deploy the contained quantum and classical programs with their dependencies. [79] proposes a framework for the execution of hybrid quantum algorithms called *algo2qpu*. Thereby, it enables the classical processing of the algorithm within the framework, e.g., the optimization of parameters for a parameterized circuit, and executes the resulting quantum circuits on cloud-based quantum computers. However, the framework does currently not support the integration with arbitrary classical programs. [80] show how a quantum program can be packaged with its dependency as a Docker container and demonstrate their approach with a prototypical implementation using Qiskit. However, their approach does not take into account the integration of the deployed quantum programs with classical programs required for the execution of hybrid quantum applications.

[9]https://cloudify.co

VII. Conclusion and Outlook

Most quantum applications are hybrid, i.e., they consist of both quantum programs and classical programs. This implies that the control- and data flow between the corresponding components, as well as their proper deployment, need to be orchestrated, and both orchestrations must be intertwined. We elucidated this by means of a real-world use case from the domain of the humanities. In order to treat hybrid quantum applications as a self-contained entity, we introduced the quantum application archive (QAA) that comprises all the artifacts of a quantum application, as well as all the required information for their processing, in a single package. The need for proven workflow technology to orchestrate the flow between the components of a quantum application, and the use of provisioning technology to orchestrate the topology of a quantum application, has been argued for. We sketched the high-level architecture of a runtime environment for quantum applications and especially revealed the role of a workflow engine and a provisioning engine in such a runtime. Furthermore, the need for a workflow modeling tool and a topology modeling tool as components of a build time environment for hybrid quantum applications have been mentioned. Finally, the sketched architecture was verified by a prototypical implementation based on the Camunda workflow system and the OpenTOSCA ecosystem.

In future work, we plan to further extend the presented prototype. Thereby, especially the integration between the queue controller on one side and the workflow engine and provisioning engine on the other side is still open. The same is true for the indicated optimization to reserve a quantum computer to a quantum application for multiple iterations, e.g., for variational algorithms. Additionally, we will investigate how the modeling of hybrid quantum applications can be supported by suited extensions of the workflow and topology modeling tools. Finally, we plan to develop workflow models implementing other quantum algorithms, such as *QAOA* [59] or *VQE* [81], and provide them in a workflow repository to enable their integration as sub-workflows into workflows solving higher-order problems.

Acknowledgment

We are very grateful for the discussions about selective subjects of this paper with our colleagues Karoline Wild and Felix Truger, as well as Daniel Fink for his support in the implementation of our sample hybrid quantum application.

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 the project *SEQUOIA* funded by the Baden-Wuerttemberg Ministry of the Economy, Labour and Housing.

References

[1] 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, 2020.

[2] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," *New Journal of Physics*, vol. 18, no. 2, 2016.

[3] 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, pp. 2–9.

[4] J. A. Cortese and T. M. Braje, "Loading Classical Data into a Quantum Computer," *arXiv:1807.02500*, 2018.

[5] R. LaRose and B. Coyle, "Robust data encodings for quantum classifiers," *Physical Review A*, vol. 102, no. 3, p. 032420, 2020.

[6] 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.

[7] L. Brenner, R. Balasubramanian, C. Burgard, W. Verkerke, G. Cowan, P. Verschuuren, and V. Croft, "Comparison of unfolding methods using RooFitUnfold," *International Journal of Modern Physics A*, vol. 35, no. 24, 2020.

[8] B. Weder, J. Barzen, F. Leymann, M. Salm, and K. Wild, "QProv: A Provenance System for Quantum Computing," *IET Quantum Communication*, 2021.

[9] 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.

[10] Y. S. Weinstein, M. Pravia, E. Fortunato, S. Lloyd, and D. G. Cory, "Implementation of the Quantum Fourier Transform," *Physical review letters*, vol. 86, no. 9, 2001.

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

[12] J. Barzen, F. Leymann, M. Falkenthal, D. Vietz, B. Weder, and K. Wild, "Relevance of Near-Term Quantum Computing in the Cloud: A Humanities Perspective," *Cloud Computing and Services Science*, vol. 1399, pp. 25–58, 2021.

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

[14] D. Vietz, J. Barzen, F. Leymann, and K. Wild, "On Decision Support for Quantum Application Developers: Categorization, Comparison, and Analysis of Existing Technologies," in *Proceedings of the 21th International Conference on Computational Science (ICCS)*. Springer, 2021, pp. 127–141.

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

[16] K. Wild, U. Breitenbücher, L. Harzenetter, F. Leymann, D. Vietz, and M. Zimmermann, "TOSCA4QC: Two Modeling Styles for TOSCA to Automate the Deployment and Orchestration of Quantum Applications," in *Proceedings of the 24th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2020, pp. 125–134.

[17] F. Leymann and J. Barzen, "Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective," *arXiv:2103.04320*, 2021.

[18] J. Barzen, "From Digital Humanities to Quantum Humanities: Potentials and Applications," in *Quantum Computing in the Arts and Humanities*. Springer, 2021, arXiv:2103.11825.

[19] F. Leymann and D. Roller, *Production Workflow: Concepts and Techniques*. Prentice Hall PTR, 2000.

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

[21] M. Wurster, U. Breitenbücher, M. Falkenthal, C. Krieger, F. Leymann, K. Saatkamp, and J. Soldani, "The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies," *Software-Intensive Cyber-Physical Systems*, 2019.

[22] T. Binz, G. Breiter, F. Leymann, and T. Spatzier, "Portable Cloud Services Using TOSCA," *IEEE Internet Computing*, vol. 16, no. 3, pp. 80–85, 2012.

[23] F. Leymann, D. Karastoyanova, and M. P. Papazoglou, "Business Process Management Standards," in *Handbook on Business Process Management 1*. Springer, 2010, pp. 513–542.

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

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

[26] J. Eder and W. Liebhart, "Workflow transactions," *Workflow Handbook*, pp. 195–202, 1997.

[27] P. Greenfield, A. Fekete, J. Jang, and D. Kuo, "Compensation is Not Enough," in *Proceedings of the 7th IEEE International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2003, pp. 232–239.

[28] J. Barzen, F. Bühler, and F. Leymann. (2021) MUSE Datenset, DaRUS, V1. [Online]. Available: https://doi.org/10.18419/darus-1805

[29] J. Barzen, M. Falkenthal, and F. Leymann, *Wenn Kostüme sprechen könnten: MUSE - Ein musterbasierter Ansatz an die vestimentäre Kommunikation im Film (in German)*. Digital Humanities. Perspektiven der Praxis, Frank und Timme, 2018, pp. 223–241.

[30] H. Jia, Y.-m. Cheung, and J. Liu, "A New Distance Metric for Unsupervised Learning of Categorical Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 5, pp. 1065–1079, 2015.

[31] Z. Wu and M. Palmer, "Verb Semantics and Lexical Selection," in *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, 1994, pp. 133–138.

[32] M. A. Cox and T. F. Cox, "Multidimensional Scaling," in *Handbook of Data Visualization*. Springer, 2008, pp. 315–347.

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

[34] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[35] J. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis, "Contemporary Web Service Discovery Mechanisms," *Journal of Web Engineering*, vol. 5, no. 3, pp. 265–290, 2006.

[36] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web services architecture," *IBM Systems Journal*, vol. 41, no. 2, pp. 170–177, 2002.

[37] S. Bowers and B. Ludäscher, "An Ontology-Driven Framework for Data Transformation in Scientific Workflows," in *International Workshop on Data Integration in the Life Sciences*. Springer, 2004, pp. 1–16.

[38] 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.

[39] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters, "Building an Encrypted and Searchable Audit Log," in *NDSS*, vol. 4. Citeseer, 2004, pp. 5–6.

[40] R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs," in *International Conference on Extending Database Technology*. Springer, 1998, pp. 467–483.

[41] M. Herschel, R. Diestelkämper, and H. Ben Lahmar, "A Survey on Provenance: What for? What Form? What from?" *The VLDB Journal*, vol. 26, no. 6, pp. 881–906, 2017.

[42] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, and J. Wettinger, "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA," in *International Conference on Cloud Engineering (IC2E)*. IEEE, 2014, pp. 87–96.

[43] R. Qasha, J. Cała, and P. Watson, "Towards Automated Workflow Deployment in the Cloud using TOSCA," in *Proceedings of the 8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 1037–1040.

[44] D. Weerasiri, M. C. Barukh, B. Benatallah, Q. Z. Sheng, and R. Ranjan, "A Taxonomy and Survey of Cloud Resource Orchestration Techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–41, 2017.

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

[46] OASIS, *TOSCA Simple Profile in YAML Version 1.3*, Organization for the Advancement of Structured Information Standards, 2020.

[47] U. Breitenbücher, T. Binz, O. Kopp, F. Leymann, and D. Schumm, "Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA," in *On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS)*. Springer, 2012, pp. 416–424.

[48] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011.

[49] B. Weder, J. Barzen, F. Leymann, and M. Salm, "Automated Quantum Hardware Selection for Quantum Workflows," *Electronics*, vol. 10, no. 8, 2021.

[50] L. Harzenetter, U. Breitenbücher, F. Leymann, K. Saatkamp, B. Weder, and M. Wurster, "Automated Generation of Management Workflows for Applications Based on Deployment Models," in *Proceedings of the 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2019, pp. 216–225.

[51] I. Odun-Ayo, B. Odede, and R. Ahuja, "Cloud Applications Management – Issues and Developments," in *International Conference on Computational Science and Its Applications (ICCSA)*. Springer, 2018, pp. 683–694.

[52] L. Harzenetter, U. Breitenbücher, K. Képes, and F. Leymann, "Freezing and Defrosting Cloud Applications: Automated Saving and Restoring of Running Applications," *Software-Intensive Cyber-Physical Systems (SICS)*, vol. 35, pp. 101–114, 2019.

[53] K. Képes, U. Breitenbücher, and F. Leymann, "Integrating IoT Devices Based on Automatically Generated Scale-Out Plans," in *Proceedings of the 10th Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2017, pp. 155–163.

[54] H. Singh and A. Sachdev, "The Quantum way of Cloud Computing," in *Proceedings of the International Conference on Reliability Optimization and Information Technology (ICROIT)*. IEEE, 2014, pp. 397–400.

[55] S. Ostermann, R. Prodan, and T. Fahringer, "Extending Grids with Cloud Resource Management for Scientific Computing," in *Proceedings of the 10th IEEE/ACM International Conference on Grid Computing*. IEEE, 2009, pp. 42–49.

[56] K. Vukojevic-Haupt, D. Karastoyanova, and F. Leymann, "On-demand Provisioning of Infrastructure, Middleware and Services for Simulation Workflows," in *Proceedings of the 6th International Conference on Service Oriented Computing and Applications (ICSOC)*. IEEE, 2013, pp. 91–98.

[57] F. Leymann, "Linked Compute Units and Linked Experiments: Using Topology and Orchestration Technology for Flexible Support of Scientific Applications," in *Software Service and Application Engineering*. Springer, 2012, pp. 71–80.

[58] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii *et al.*, "Variational Quantum Algorithms," *arXiv:2012.09265*, 2020.

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

[60] Camunda. (2021) Camunda BPMN Workflow Engine. [Online]. Available: https://camunda.com/products/camunda-bpm/bpmn-engine

[61] Camunda. (2021) Camunda BPMN Modeler. [Online]. Available: https://camunda.com/products/camunda-bpm/modeler

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

[63] U. Breitenbücher, C. Endres, K. Képes, O. Kopp, F. Leymann, S. Wagner, J. Wettinger, and M. Zimmermann, "The OpenTOSCA Ecosystem - Concepts & Tools," *European Space project on Smart Systems, Big Data, Future Internet-Towards Serving the Grand Societal Challenges*, vol. 1, pp. 112–130, 2016.

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

[65] M. Zimmermann, U. Breitenbücher, L. Harzenetter, F. Leymann, and V. Yussupov, "Self-Contained Service Deployment Packages," in *Proceedings of the 10th International Conference on Cloud Computing and Services Science (CLOSER)*. SciTePress, 2020, pp. 371–381.

[66] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "A framework for QoS-aware binding and re-binding of composite web services," *Journal of Systems and Software*, vol. 81, no. 10, pp. 1754–1769, 2008.

[67] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and Entangling Capability of Parameterized Quantum Circuits for Hybrid Quantum-Classical Algorithms," *Advanced Quantum Technologies*, vol. 2, no. 12, 2019.

[68] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, "Parameterized quantum circuits as machine learning models," *Quantum Science and Technology*, vol. 4, no. 4, 2019.

[69] A. J. Hey, S. Tansley, and K. M. Tolle, *The Fourth Paradigm – Data-intensive Scientific Discovery*. Microsoft Research, Redmond, WA, 2009, vol. 1.

[70] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A Survey of Data-Intensive Scientific Workflow Management," *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[71] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. R. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," *Nucleic Acids Research*, vol. 34, no. suppl_2, pp. W729–W732, 2006.

[72] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[73] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and computation: Practice and experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[74] 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.

[75] J. T. Krogel, "Nexus: A modular workflow management system for quantum simulation codes," *Computer Physics Communications*, vol. 198, pp. 154–168, 2016.

[76] Zapata. (2021) Orquestra. [Online]. Available: https://www.zapatacomputing.com/orquestra

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

[78] R. Qasha, J. Cała, and P. Watson, "A Framework for Scientific Workflow Reproducibility in the Cloud," in *Proceedings of the 12th IEEE International Conference on e-Science (e-Science)*. IEEE, 2016, pp. 81–90.

[79] S. Sim, Y. Cao, J. Romero, P. D. Johnson, and A. Aspuru-Guzik, "A framework for algorithm deployment on cloud-based quantum computers," *arXiv:1810.10576*, 2018.

[80] P. Dreher and M. Ramasami, "Prototype Container-Based Platform for Extreme Quantum Computing Algorithm Development," in *Proceedings of the 23rd IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2019, pp. 1–7.

[81] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

All links were last followed on November 12, 2021.