Institute of Architecture of Application Systems

TOSCA4QC: Two Modeling Styles for TOSCA to Automate the Deployment and Orchestration of Quantum Applications

Karoline Wild, Uwe Breitenbücher, Lukas Harzenetter, Frank Leymann, Daniel Vietz, and Michael Zimmermann

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

BibTeX:

```
@inproceedings{Wild2020_TOSCA4QC,
  author    = {Wild, Karoline; Breitenb{\"u}cher, Uwe; Harzenetter, Lukas;
               Leymann, Frank; Vietz, Daniel; Zimmermann; Michael},
  title     = {{TOSCA4QC: Two Modeling Styles for TOSCA to Automate the
               Deployment and Orchestration of Quantum Applications}},
  booktitle = {Proceedings of the 24th International Enterprise Distributed
               Object Computing Conference (EDOC 2020)},
  publisher = {IEEE},
  year      = 2020,
  month     = oct,
  pages     = {125--134},
  doi       = {10.1109/EDOC49727.2020.00024}
}
```

© 2020 IEEE Computer Society. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Universität Stuttgart
Germany

TOSCA4QC: Two Modeling Styles for TOSCA to Automate the Deployment and Orchestration of Quantum Applications

Karoline Wild, Uwe Breitenbücher, Lukas Harzenetter, Frank Leymann, Daniel Vietz, and Michael Zimmermann
Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{wild, breitenbuecher, harzenetter, leymann, vietz, zimmermann}@iaas.uni-stuttgart.de

***Abstract*—Quantum computing introduces a new computing paradigm that promises to solve problems that cannot be solved by classical computers efficiently. Thus, quantum applications will be more and more integrated in classical applications. To bring these composite applications into production, technologies for an automated deployment and orchestration are required to avoid manual error-prone and time-consuming processes. For non-quantum applications, a variety of deployment technologies have been developed in recent years. However, the deployment of quantum applications currently differs significantly from non-quantum applications and thus, leads to a different modeling procedure for the deployment of quantum applications. To overcome these problems, we propose TOSCA4QC that introduces two deployment modeling styles based on the Topology and Orchestration Specification for Cloud Applications (TOSCA) standard for automating the deployment and orchestration of quantum applications: (i) SDK-specific modeling style to cover all technical deployment details and (ii) SDK-agnostic modeling style supporting common modeling principles. We further show how existing model-driven development (MDD) approach can be applied to refine a SDK-agnostic model to an executable SDK-specific model. We demonstrate the practical feasibility by a prototypical implementation as an extension of the TOSCA ecosystem OpenTOSCA and three case studies with IBMQ and a quantum simulator.**

***Index Terms*—TOSCA, Quantum Computing, Deployment Automation, Modeling, Orchestration**

I. Introduction

Quantum computing introduced a completely new computing paradigm. By exploiting quantum mechanical effects such as superposition and entanglement, quantum computing promises to solve problems that cannot be solved or not solved efficiently with conventional computers [1]–[3]. Since quantum computing offers an advantage especially for certain problem classes, quantum applications will be more and more integrated in classical applications. However, this integration also requires deployment and orchestration automation to avoid time-consuming and error-prone manual processes [4].

For deployment and orchestration automation, a variety of technologies have been developed in the last years. Thereby declarative deployment technologies, such as Terraform, Kubernetes, or the Topology and Orchestration Specification for Cloud Applications (TOSCA) standard, have prevailed [5]. A declarative deployment model enables the specification of application deployments by modeling the application's structure consisting of its components and their relations [6]. These technologies are typically extensible by providing features to support deploying new component types on new target infrastructures. Especially the TOSCA standard has shown to be highly extensible: Beyond cloud computing, it has been adopted also in other areas [7], e. g., Internet of Things [8]–[10] and Network Function Virtualization Orchestration [11], [12].

However, the current state-of-the-art of available quantum platforms and providers introduces some special characteristics regarding the deployment of quantum applications that violate the principles supported by common deployment technologies. First, quantum applications must be newly deployed for each invocation in contrast to classical applications, which have to be deployed only once and can be invoked multiple times afterwards. Second, an application's code is typically hosted on the environment it is actually executed on, as it is, e. g., known from existing cloud service models such as Infrastructure-as-a-Service (IaaS). In contrast, currently hosting a quantum application's code is not supported by today's *Quantum-as-a-Service (QaaS)* offerings. Therefore, the code of a quantum application must be hosted on an external, conventional computing resource, where a software development kit is available that is capable of compiling the quantum algorithm and managing its actual deployment on a selected quantum computer during runtime. Consequently, extending existing deployment technologies to support the deployment and orchestration of quantum applications requires special consideration and, without special consideration, leads to a modeling process that is not consistent with established modeling concepts known from non-quantum applications.

To meet these challenges, we exploit the extensibility of TOSCA and introduce *TOSCA4QC* that defines two *deployment modeling styles* to automate the deployment and orchestration of quantum applications: (i) the *SDK-specific (SDK-S)* modeling style, which covers all technical details and (ii) the *SDK-agnostic (SDK-A)* modeling style, which reflects common modeling principles known from classical applications while hiding technical details. We further show how a model-driven development (MDD) approach [13] can be used to refine a SDK-A model to an executable SDK-S model. To proof the practical feasibility the open-source TOSCA ecosystem OpenTOSCA [14] has been extended and three case studies based on IBMQ and a quantum simulator has been developed.

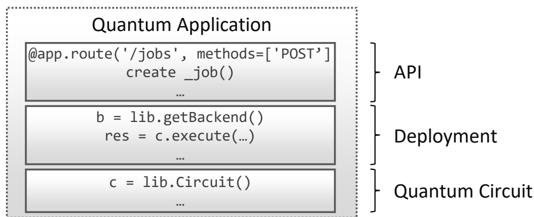

Figure 1. Components of a quantum application

Table I
EXCERPT OF AVAILABLE QUANTUM SDKS, THEIR PROGRAMMING LANGUAGES, AND SUPPORTED QUANTUM CLOUD PROVIDERS.

SDK	Progr. Language	Cloud Providers		
		IBMQ	*Rigetti*	*Azure**
Qiskit	Qiskit (Python)	x		
Forest	pyQuil (Python)		x	
QDK	Q#			x
XACC	XACC (C++, Python)	x	x	
Pennylane	Pennylane (Python)	x	x	(x)

II. FUNDAMENTALS, MOTIVATION, AND PROBLEM STATEMENT

In this section we first introduce the fundamentals of quantum computing, quantum providers, and quantum applications. We then introduce two key principles of deployment technologies, which are based on the characteristics of classical applications. Based on that, the problem statements are summarized. Finally, the key elements of the TOSCA standard are briefly introduced.

A. Quantum Computing in the NISQ Era

Quantum computing is a completely new computing paradigm, and the quantum computers are in an early stage. Similar to bits in conventional computers, quantum computers consist of so-called *quantum bits (qubits)* [15]. While bits are either in state 0 or 1, qubits can be in both states at the same time, called superposition [15], [16]. Superposition, which implies quantum parallelism, and the entaglement of qubits are the drivers behind the speedup compared to conventional computers. However, current generations of quantum computers are limited with regard to several properties [2]: (i) The number of qubits is limited and, thus, the calculable problem size is restricted, (ii) operations on qubits are inaccurate, and (iii) qubits can hold their state stable only for a limited amount of time, i. e., they are “noisy”. Due to these limitations, quantum computers in this era are often called *Noisy Intermediate Scale Quantum (NISQ)* computers [2].

For the quantum computation itself there are currently various quantum computation models, such as gate-based [17], measurement-based [18], or adiabatic [19]. In the gate-based model, on which we focus in this paper, operations are represented as *gates* which are applied to qubits. To retrieve the result of a computation, the values of the qubits are measured and interpreted. A sequence of gates applied on qubits and their measurement form a so-called *quantum circuit* [15]. To compile, execute, and integrate a quantum circuit into a non-quantum application, additional logic is required, which is discussed in the next section.

B. Quantum Application and Quantum Service Provider

For implementing and executing quantum circuits, several software development kits (SDKs) exist that provide a set of libraries and tools. In Table I, an excerpt of existing SDKs with their supported programming languages and quantum cloud offerings is shown. All programming languages are embedded in existing languages such as Python or C++, only Q# is a new one tailored to quantum algorithms. While IBMQ and Rigetti already provide publicly available QaaS offerings, Azure only announced an upcoming service. Nevertheless, they already provide a SDK called Quantum Development Kit (QDK) [20] which can be used for simulating quantum algorithms. These SDKs include compilers to translate the quantum circuit into quantum computing languages and connectors to cloud providers and their QaaS offerings [17]. Thus, the quantum circuit itself is just a part of a *quantum application* as depicted in Figure 1. Methods to select a quantum computer, compile and execute the circuit, and an API to enable the invocation of the quantum algorithm are essential parts of today’s quantum applications. Thus, the code of a quantum application must be persistently hosted on a conventional compute resource where the respective SDK is installed, but the compiled circuit is actually executed on a quantum computer.

As shown in Table I, two different kinds of SDKs exists: On the one hand, Qiskit [21], Forest [22], and QDK which are provider-specific SDKs designed to meet the needs of the quantum computers made available by the quantum computer vendors themselves through the cloud. On the other hand, third-party provider-independent SDKs that potentially support multiple quantum providers, such as XACC [23] and Pennylane [24], have emerged. XACC supports two gate-based providers, IBMQ and Rigetti, but also D-wave which supports an adiabatic quantum computation model. Also Pennylane supports IBMQ and Rigetti, but also a photonic quantum computer and the QDK simulator. Thus, in order to implement and run a quantum application, such a SDK must currently be installed on a conventional computer to host the actual code, compile the quantum circuit, and to run it on the selected quantum computer. Cloud providers, such as AWS, announced to offer quantum computers from various vendors via the cloud [25]. Hence, new service delivery models could be available soon, which may be able to operate without external resources to execute a quantum circuit.

C. Deployment Principles

For automating the deployment and orchestration of classical applications a variety of deployment technologies exist. In general two deployment modeling approach can be distinguished: *declarative* and *imperative* deployment modeling technologies [6]. While declarative models describe the structure of applications with their components and relations, imperative models explicitly describe the operations and their order in which they have to be executed to deploy an application. In

practice, declarative deployment technologies have become established [5], [26]. Among the widely used technologies that follow the declarative modeling approach are Chef, Terraform, CloudFormation, Kubernetes, but also the TOSCA standard, which is described in more detail in Section II-E. In previous work, we investigated 13 declarative deployment technologies to analyze their common modeling meta model [5]. In this work, we continued our analysis by deriving two basis *deployment principles* of classical applications that are supported by all 13 declarative deployment technologies and that determine the provisioning order of the application's components [27]: (i) *Host-and-Execute* and (ii) *Deploy-Before-Invoke*. Both principles are described in detail below.

Figure 2 depicts a declarative deployment model of a classical application that consists of two application components, Web App and Order Processor. The WebApp is hosted on AWS Elastic Beanstalk and the Order Processor on an Ubuntu virtual machine where Java in installed. In both cases is the hosting environment equivalent to the execution environment. Thus, the first deployment principle is as follows:

Deployment Principle (DP) 1 *Host-Executes-Application: "If the deployment model specifies that an application is hosted on another component, this other component executes the application."*

Moreover, the Web App has to be connected to the Order Processor. However, the Web App can only connect to the Order Processor if it *exists*, i. e., if the Order Processor component is deployed in advance. Thus, the second deployment principle is as follows:

Deployment Principle (DP) 2 *Deploy-Before-Invoke: " If the deployment model specifies that an application component invokes another component, the invoked component must be deployed before the invocation can take place, i. e., before the deployment technology establishes the connection between the components."*

These principles are driven by the "nature" of classical applications and compute resources. They form the key principles every declarative deployment technology is following and which specify a common understanding of the deployment order of classical application components.

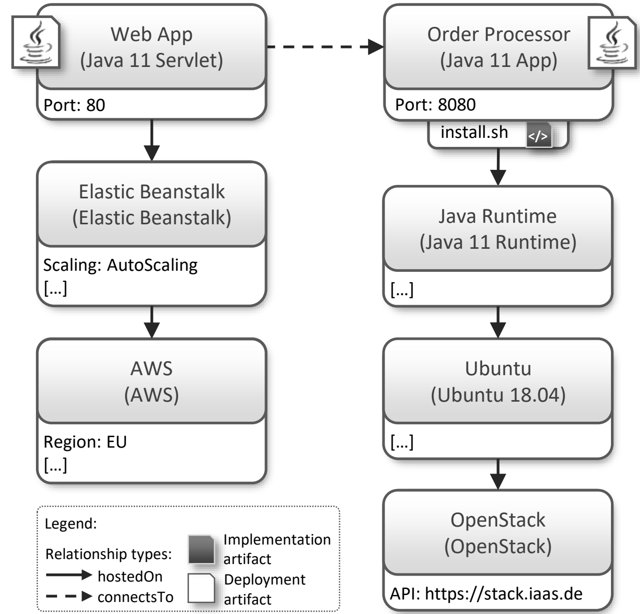

Figure 2. Exemplary Deployment Model (notation adapted from [28])

D. Problem Statements and Research Questions

The two deployment principles described in the previous section are violated by current quantum SDKs and QaaS offerings. First, the code of the quantum application including the quantum circuit has to be persistently hosted on external conventional compute resources, but the quantum circuit is executed by a quantum computer and not by the conventional computing resource on which it is hosted (DP1). Second, for each invocation the quantum circuit has to be compiled depending on the input data [29], and then the compiled quantum circuit is sent to and executed by the selected quantum computer provided by a cloud services (DP2). The compilation, deployment, and execution of the quantum circuit is handled by a SDK, such as Qiskit, deployed on the external compute resource. As a result compilation, deployment, and execution of quantum circuits cannot be separated yet and are executed during runtime for each invocation of the quantum application. Thus, the first research question (RQ) is as follows:

RQ 1 *"How can the deployment and orchestration of quantum applications be automated if deployment and invocation cannot be separated?"*

The deployment of quantum applications not only violates the two key deployment principles but also requires in-depth knowledge about the quantum computers, the quantum providers, and the SDKs [29]. Depending on the used programming language or libraries for implementing the quantum application only a specific SDK can be used and, thus, the set of available quantum computers is reduced [30]. Especially for modelers who are not familiar with the technical details of quantum applications and focus on the integration with non-quantum applications, it is time-consuming and error-prone to orchestrate all technical details for the deployment correctly. Moreover, a holistic modeling approach that follows the same modeling principles for both, quantum and non-quantum applications, enables the modeling of composite applications. Thus, the second research question is as follows:

RQ 2 *"How can the deployment and orchestration of quantum applications be modeled in a way that reflects common modeling principles?"*

E. The Topology and Orchestration Specification for Cloud Applications (TOSCA) Standard

Among the many domain-specific languages used by various deployment technologies, TOSCA is the only standardized modeling language [31], [32]. It is an OASIS standard originally developed for defining deployment models for automating the deployment and orchestration of cloud applications. The central element in TOSCA is the *service template* which comprises (i) the structure of the application in the form of a *topology template* and (ii) *management plans* that encompass the knowledge about managing the application described as workflows (e. g., using BPEL or BPMN). Since the structural description is the primary focus of this paper, we concentrate on the topology template. In Figure 2 an example of a TOSCA deployment model is shown. To improve readability, Figure 2 uses a graphical notation for TOSCA instead of XML or YAML, for which TOSCA is actual available.

A topology template is a directed acyclic graph (DAG) that describes the structure of an application with its components, i. e., nodes, as *node templates* and their relations, i. e., edges, as *relationship templates*. This includes application-specific components as well as middleware and infrastructure components. The application shown in Figure 2 is part of an order application consisting of a web application component, called Web App, and a processing component, called Order Processor. The Web App shall be hosted on AWS Elastic Beanstalk, while the Order Processor will be hosted on an Ubuntu virtual machine on-premise. The semantics of node templates, as well as relationship templates, are defined by reusable *node types* and *relationship types*. The TOSCA standard specifies normative types such as the used relationship types “hostedOn” and “connectsTo” in Figure 2 or node types such as “Compute” from which other node types such as “Ubuntu 18.04” are derived. The hostedOn dependencies form the entire stack of an application while connectsTo expresses that a communication relation must be established. With defined *properties* the configuration of node and relationship templates can be specified, e.g., the port number of the Web App or the region of the AWS data center.

For the actual deployment and management of an instance of a service template both, node types and relationship types, define lifecycle operations to implement the behaviour of the orchestration engine. Normative lifecycle operations are create, configure, start, stop, and delete. The actual executable file implementing either an operation or a node template itself is called *implementation artifact (IA)* if it implements an operation, or *deployment artifact (DA)* if it implements a node template. TOSCA provides also a strong typing for artifacts: *Artifact types* and *artifact templates* that can be added to node templates. In Figure 2, the *install.sh* IA for the install-operation of the Order Processor component and two WAR files as DAs for the Web App and the Order Processor are shown as examples. The service template and all artifacts are then packaged in a self-contained *cloud service archive (CSAR)* which can then be processed by any TOSCA-compliant runtime, e.g., the open-source OpenTOSCA container [14], [33].

III. TOSCA4QC: TOSCA Modeling Styles for Quantum Applications

In this section we introduce *TOSCA4QC* for modeling the deployment and orchestration of quantum applications. For this two *deployment modeling styles* with TOSCA have been introduced. Since TOSCA provides an ontological typing mechanism, i. e., new types can be defined without extending the language definition itself [34], the TOSCA language definition has not been adapted for the deployment modeling styles. Before the two modeling styles are introduced, general aspects are discussed in context of the deployment principles in Section II-C and the research questions in Section II-D.

A. TOSCA4QC Deployment Modeling Styles Overview

A key aspect of the modeling styles is to be compliant with the deployment principles of classical applications, while the resulting models are still deployable despite the non-separability of deployment and invocation of quantum applications. Since if the deployment principles are violated the deployment mechanics of the deployment engines that process the declarative deployment models also are violated. An important requirement of the modeling styles is therefore that they are conform to the TOSCA standard and the models can therefore be processed by any TOSCA-compliant deployment engine. A second challenge is the diversity of stakeholders in the quantum computing domain in terms of their areas of expertise and technical know-how. In previous works, different stakeholders have already been identified [35]–[37]. On the one hand, there are quantum experts that have in-depth knowledge of existing quantum algorithms, quantum hardware, and they can implement these algorithms for specific quantum computers [29]. On the other hand, there are software developers who only want to integrate quantum applications to solve specific problems efficiently.

Therefore, in addition to a modeling style that results in a technically detailed and executable model, which reflects all SDK-specific properties, but which does not follows common modeling principles (RQ1), we present a simplified modeling style, but which is not compliant with the deployment principles and therefore a resulting model is not executable by a TOSCA-compliant deployment engine (RQ2), and has to be transformed before deployment, as described in Section IV. The first style, the *SDK-specific (SDK-S)* modeling style, which supports a detailed and fine-grained modularization of node types and that specify all technical details, such as the different SDKs and QaaS, is introduced in Section III-B. A SDK-S model can directly be processed by each TOSCA-compliant deployment engine. To make this possible, however, it is not possible to model where the quantum circuit is actually executed, and especially for non-quantum experts this differ from their known modeling approach. Therefore, an additional modeling style, the *SDK-agnostic (SDK-A)* modeling style has been defined and is introduced in Section III-C. A SDK-A model hides the technical details and enables a simplified modeling for modelers that are not familiar with all technical details of quantum computing. However, because the technical details are hidden, a SDK-A deployment model cannot directly be

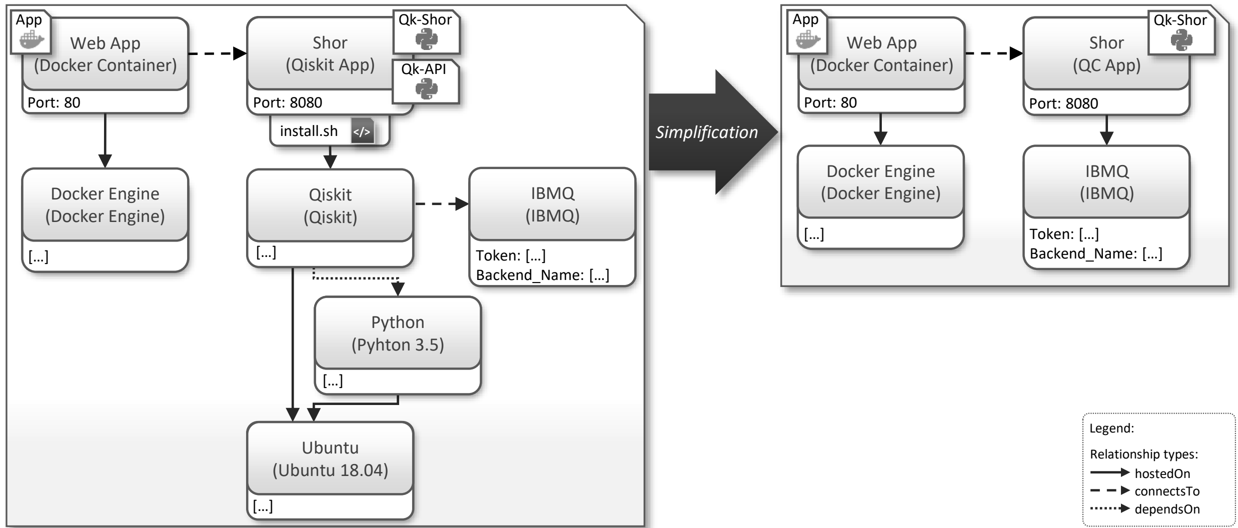

Figure 3. TOSCA4QC modeling styles: (a) *SDK-specific (SDK-S)* modeling style on the left and (b) *SDK-agnostic (SDK-A)* modeling style on the right.

executed by a TOSCA-compliant deployment engine. Thus, in Section IV we introduce a model-driven development (MDD) approach to refine a SDK-A deployment model to an executable SDK-S deployment model in an automated manner.

B. SDK-Specific Modeling Style

In this section the *SDK-specific (SDK-S)* modeling style with its characteristics, advantages, and drawbacks is introduced.

1) Characteristics: The SDK-S modeling style defines a general structure with general required component types: (i) A *SDK-specific quantum application* node that represents the quantum application itself and that is specific for the used SDK, e. g., a *Qiskit App* or *Pennylane App* node type has be be defined to encapsulate the deployment and invocation logic, since they cannot be separated yet as well as the circuit developed with the respective SDK. (ii) A *SDK* node that represents the SDK required to compile and execute the quantum circuit which is part of the quantum application, e. g., a *Qiskit* or *Pennylane* node type has to be defined to install the required packages on a defined compute resource. (iii) A *compute* resource to host the SDK and the application, e. g., an *Ubuntu 18.04* or any other conventional compute resource. (iv) A *quantum compute* node that represents the execution environment for the quantum circuit, e. g., a *IBMQ* node type, the SDK can connect to.

In Figure 3 on the left, an exemplary SDK-S deployment model with a quantum application that is invoked by a web application is shown. The Web App is a *Docker Container* that is *hostedOn* a *Docker Engine* and that has a *connectsTo* relationship to the quantum application. The quantum application is a *Qiskit App*. In the example, *Shor*, an algorithm for factorizing numbers in polynomial time [38], has to be deployed. The Qiskit App node type encompasses not only a DA (*Qk-Shor*) containing the quantum algorithm, i. e., in this example a Python file implementing the Shor algorithm using the Qiskit SDK, but also a DA (*Qk-API*) that provides its invocation and deployment logic. Since the invocation and execution logic is independent of the algorithm, this DA is reusable for all Qiskit implementations. The execution of the quantum circuit will be performed by IBMQ. The *IBMQ* node type represents the IBMQ cloud services and is used to configure the execution of the quantum circuit on IBMQ. This includes a “Token” required for the authentication at the cloud service’s API and a “Backend_Name” of the selected quantum computer that executes the quantum circuit. The *Qiskit* node requires these inputs to establish a connection to IBMQ and to invoke the service during runtime with the compiled circuit. Qiskit is an open-source SDK that requires a *Python 3.5* environment and must be hosted by an arbitrary compute resource that supports Python and Qiskit. In this example an *Ubuntu 18.04* is used.

2) Advantages: The fine-grained modularization facilitates a detailed configuration. The modelling style is *technically* compliant with both deployment principles: The actual hosting relations, namely that the quantum application is hosted on the conventional compute resource, are reflected (DP1) and since invocation and deployment logic cannot be separate yet, it is encapsulated in the API DA of the quantum application, i. e., in this scenario the API receives the invocation calls and triggers the deployment of the quantum circuit (DP2). Thus, it is compliant with the deployment mechanics of TOSCA and can be processed by each TOSCA-compliant deployment engine.

3) Drawbacks: However, by following the principles, the resulting model implies that nothing is executed on IBMQ and this is actually not true. One reason for this is that the quantum circuit cannot be separated from the remaining quantum application and thus it is not a separate, executable component in the deployment model. This makes it difficult for non-experts to understand and apply this modeling style.

C. SDK-Agnostic Modeling Style

To overcome the drawbacks of the SDK-S modeling style and to provide an modeling approach that reflects common modeling principles, in this section the *SDK-agnostic (SDK-agnostic)*

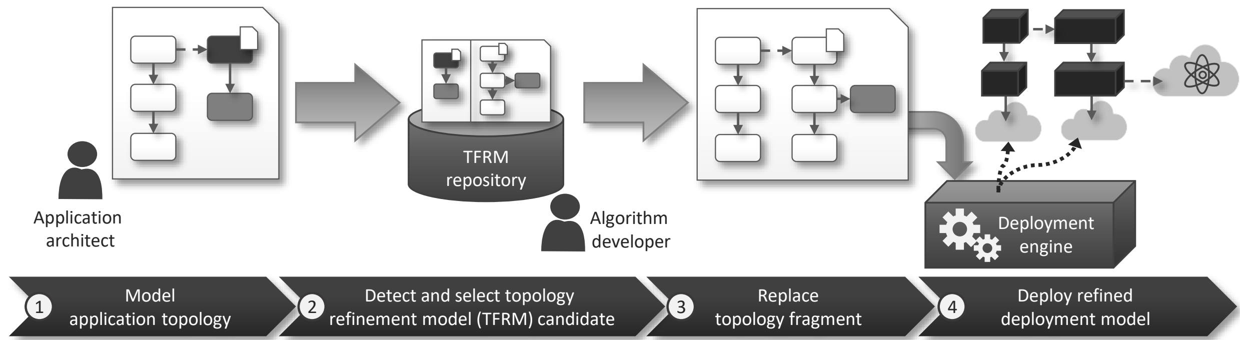

Figure 4. Topology Refinement Process

modeling style with its characteristics, advantages, and drawbacks is introduced.

1) Characteristics: The general structure of SDK-A deployment models consists only of two components: (i) A *quantum application*, which in contrast to the quantum application in the SDK-S style is agnostic to the SDK used. Therefore, only one general *QC App* node type is required to define the quantum application and arbitrary DAs can be attached, e. g., a Qiskit implementation. (ii) A *quantum compute* node that is equivalent to the quantum compute node in the SDK-S style. It represents the execution environment for the quantum circuit and can be, e. g., an *IBMQ* node type. Figure 3 on the right depicts a SDK-A model which is a simplification of the SDK-S model on the left. While the non-quantum stack with the Web App remains the same, the quantum application stack is notably simplified. In contrast to the SDK-S model, details such as the required SDK are hidden. The *QC App* node type represents any quantum application, in this example the Shor algorithm implemented with Qiskit is attached, that is *hostedOn* a QaaS. In this example, *IBMQ* is used as execution environment.

2) Advantages: The SDK-S modeling style enables the modeling of the deployment of quantum applications in the same way as non-quantum applications: The quantum application representing the quantum circuit to be executed is hosted on the quantum compute node where it is actually executed.

3) Drawbacks: However, in SDK-A models both deployment principles are violated, since the quantum application cannot be hosted on a quantum compute node and thus it can also not be invoked. Thus, the deployment mechanics of TOSCA are violated and the model cannot be processed. To solve this problem for current services, a MDD approach to refine a SDK-A deployment model to an executable SDK-S deployment model is presented in the following.

IV. Model Refinement and Deployment Automation

The proposed *topology refinement* approach follows the concept of MDA and enables refining SDK-A deployment models into executable SDK-S deployment models in an automated manner. As soon as offerings such as IBMQ support to host and integrate quantum circuits into applications by providing invocation mechanisms, the SDK-A style reflects exactly this hosting relation, and then, if these two features are available, both deployment principles holds and a transformation is not needed anymore. However, for current cloud offerings, the refinement approach is required to enable both (i) a modeling approach that reflects common modeling principles and (ii) compliance with the deployment principles of TOSCA. For this, we extended the model refinement approach introduced in previous works [39]–[41]. In the following, first an overview of the model refinement process is given and then the refinement mechanism is explained in detail.

A. Model Refinement Process Overview

The model refinement process is shown in Figure 4. It is adopted and extended from the pattern refinement approach introduced by Harzenetter et al. [39]. First, the application topology can be modeled as described in Section III-C (step 1): A QC App that is hosted on a QaaS depicted in the abstracted topology with two dark grey node templates. The left stack, depicted as white node templates, represents an arbitrary non-quantum application that invokes the functionality provided by the quantum application. The implementation of the quantum application can also be attached as DA to the QC App.

To enable the deployment of the quantum application stack, the model must be refined with the SDK, e. g., Qiskit, and an operating system, e. g., a virtual machine, hosting the SDK and the quantum algorithm. However, the refinement depends on (i) the selected QaaS, (ii) the used SDK for the development of the quantum algorithm, and (iii) the programming language. Since a manual refinement is error-prone and time-consuming, we use an automated approach which identifies the SDK-A fragment in a topology template that must be refined by a SDK-S fragment. We hereby use so called *topology fragment refinement models (TFRMs)*, which are introduced in detail in the next section. These TFRMs are provided in a repository (step 2). Based on a *detector* that describes a SDK-A fragment, TFRMs are identified as suitable refinement candidates. The so detected SDK-A fragment in a topology can then be replaced by the *refinement structure* defined by the TFRM (step 3). The refinement structure is a SDK-S fragment with all technical details required for a fully automated deployment of the quantum application. In case multiple TFRMs are detected, the modeler can select the preferred one which is then applied to the deployment model. Then it can be processed by a TOSCA-

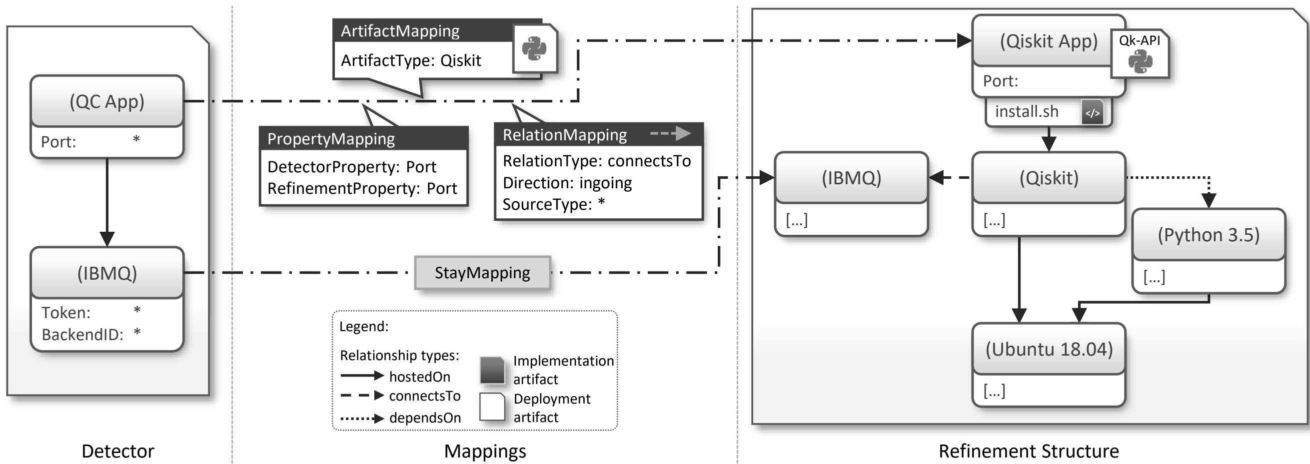

Figure 5. Topology Fragment Refinement Model

compliant deployment engine that instantiates the application as defined by the topology template (step 4).

The next section describes the TFRM in more detail. The refinement approach is not only applicable to quantum applications, but to all applications where reduced model complexity supports the modeling process and deployment details are only relevant at deployment time. A similar mechanism has been applied to process modeling [42].

B. Topology Fragment Refinement Model

To automate the refinement of SDK-A models, we introduce *topology fragment refinement models (TFRMs)* which describe how a topology template fragment which is not deployable can be refined to concrete components and technologies that define detailed configuration properties. TFRMs are a further development of the pattern refinement models (PRMs) already introduced by Harzenetter et al [39], [41].

A TFRM consists of (i) a topology template called *detector*, (ii) a topology template called *refinement structure*, and (iii) a set of *mappings* that define the relation between the elements of the detector and the refinement structure. An example TFRM is shown in Figure 5. It illustrates how a SDK-A fragment can be refined to an executable SDK-S fragment in an automated manner. The detector of a TFRM defines a topology template that can be refined by the specified TFRM. The *detector* identifies refinable fragments in a topology template based on the used node and relationship types. In Figure 5, the TFRM is *applicable* if a node template of type *QC App* is *hostedOn* a node template of type *IBMQ*. The properties are hereby not part of the detector and, thus, the *asterisk* is used as wild card to indicate that arbitrary content can be placed here.

A fragment of a topology template matching a detector can be replaced with the TFRM's *refinement structure*. In Figure 5, the topology fragment defined in the detector can be refined to the topology fragment shown on the right side. This includes the node types, relationship types, configuration properties, as well as IAs and DAs. Hereby, the refinement structure corresponds to the topology template shown in Figure 3 on the left. It enables the deployment of a quantum application containing a quantum algorithm implemented with Qiskit. In addition to the structure being refined, it is important that (i) the refinement structure is inserted correctly into the remaining topology template and (ii) existing information is not lost. To ensure that the fragment is inserted correctly and that important information is preserved, *mappings* are defined. We distinguish four mapping types, whereby two are new contributions:

- *Relation Mappings* [39] specify how ingoing or outgoing relationship templates of the detected fragment must be redirected to the refined fragment. The relation mapping in Figure 5 defines that all *connectsTo* relationship templates ingoing at the node template of type *QC App* must be redirected to the node template of type *Qiskit App*.
- *Artifact Mappings* (new) specify which DA attached to a node template of the detected fragment is moved to which node template of the refined fragment. In this example, if a DA of type *Qiskit* is attached to the node template of type *QC App*, it is moved to the node template of type *Qiskit App* in the refined fragment.
- *Property Mappings* (new) specify which property of a node template in the detected fragment corresponds to which property of a refinement structure node template. Thus, if already specified, the value of the property can be transferred. In the example, the value of the *Port* property of the node template of type QC App is added to the *Port* property of the node template of type Qiskit App.
- *Stay Mappings* [41] define that a node template of the detected fragment is equivalent to one of the refined fragment. Thus, this element remains unchanged. In the example, the node template of type *IBMQ* is unchanged.

While property and stay mappings refer to elements in the detector and refinement structure, relation and artifact mappings refer to elements that are neither in the detector nor the refinement structure. Thus, if and only if there is an artifact attached to a node template, or an ingoing or outgoing relationship template, the artifact or relationship template must match the defined

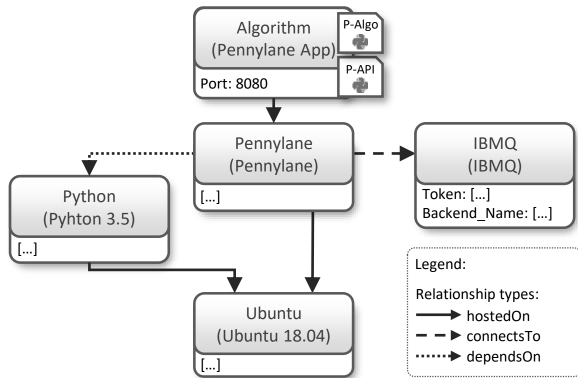

Figure 6. Exemplary topology template for a Pennylane algorithm on IBMQ.

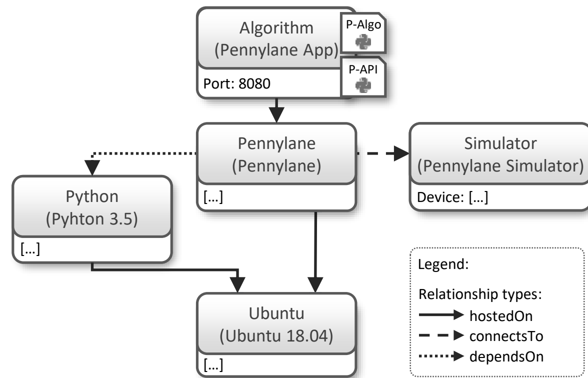

Figure 7. Exemplary topology template for a Pennylane algorithm on the local Pennylane simulator.

mapping, otherwise the TFRM is not applicable. Transferred to the TFRM example, this means that if there is an ingoing relationship template at the QC App node template that is not of type *connectsTo*, the relationship template cannot be redirected and therefore the TFRM cannot be applied. Similarly, if an artifact is attached to the QC App node template which is not of type *Qiskit* the TFRM is also not applicable. If no relationship or artifact is used, the mappings are ignored. Thus, it is ensured that the refined deployment model is deployable.

V. Prototype

We used the open-source TOSCA-compliant ecosystem OpenTOSCA [14], [43] to model, refine, and execute the quantum application deployment models. It consists of the modeling tool *Winery*, the deployment engine *OpenTOSCA Container*, and the self-service portal *Vinothek*. Winery provides capabilities (i) to manage all TOSCA elements, including service templates, node types, and relationship types, and (ii) to model topology templates graphically. In order to support TFRMs, Winery has been extended to enable their creation, management, as well as their automated application. Only the two new mappings had to be added to the existing refinement mappings. To create a new TFRM, the detector and refinement structure can be modeled graphically as topology templates. Instead of modeling these topology template fragments from scratch, an existing topology template can be imported. Thus, solutions that have been implemented and tested can be directly reused as refinement structures.

The mappings can be defined based on the elements contained in the detector and refinement structure (property and stay mappings) or other elements available in Winery, e. g., artifact types and relationship types (relation and artifact mapping). The TFRMs are globally available for their application to corresponding topology templates. Before the topology template can be packaged and processed by the OpenTOSCA Container, the refinement process is applied. If there are multiple quantum applications modeled in the topology template, the refinement process is repeated until all SDK-A fragments are refined with suitable executable SDK-S fragments. The prototypical implementation is publicly available [44].

VI. Case Studies

To demonstrate the practical feasibility of the proposed modeling styles, we implemented three SDK-S deployment models and the respective three TFRMs. Due to the limited free-to-use quantum cloud services, only IBMQ has been integrated to execute quantum algorithms on quantum computers. We further integrated the Pennylane simulator to demonstrate the flexibility of the SDK-S modeling style and its generality. We realized the following three use cases as deployable CSARs: (1) *Qiskit with IBMQ*, the running example already explained in detail in the previous sections with Shor as exemplary algorithm, (2) *Pennylane with IBMQ* with an algorithm developed with Pennylane and executed on IBMQ, depicted in Figure 6, and (3) *Pennylane with simulator* with an algorithm developed with Pennylane and executed on the local Pennylane simulator, depicted in Figure 7. Since use case (1) is already described in detail in the previous sections use case (2) and (3) are explained in detail in the following[1].

Pennylane [24] is a vendor-independent SDK to develop and execute quantum applications and is specifically designed to work with machine learning libraries and different quantum cloud services. Running an algorithm on a particular QaaS also requires the SDK, such as Qiskit, that manages the access. However, Pennylane can be installed together with all quantum cloud services or simulators it supports. Thus, depending on the selected backend, Pennylane will be configured accordingly during the deployment. In Figure 6 and Figure 7 the two SDK-S deployment models with Pennylane using IBMQ and the Pennylane simulator for executing the quantum circuit are depicted. The node type *Pennylane App* represents the quantum application that contains a quantum algorithm implemented with Pennylane. To compile and execute the quantum algorithm Pennylane is also required and, thus, the node type *Pennylane* is defined and the Algorithm node template is hosted on Pennylane. As execution environment in Figure 6, IBMQ is selected, thus, Pennylane *connectsTo* IBMQ which is reused

[1] The respective service templates as well as topology fragment refinement models can be found in the OpenTOSCA TOSCA definitions repository: https://github.com/OpenTOSCA/tosca-definitions-public

from the Qiskit use case. In Figure 7 the Pennylane simulator is used as execution environment. Even though it is actually installed with Pennylane, to enable a flexible and universal modeling, an execution environment is always modeled as individual node template. Similar to Qiskit, Pennylane requires a Python environment to be installed on the hosting machine for which again an Ubuntu virtual machine is used.

Modeling the same use case as SDK-A deployment model results in a similar topology template as shown in Figure 3 on the right. Only the *artifact type* of the attached DA would be different. Instead of *Qiskit* it has to be of type *Pennylane*. Because of the different artifact type another refinement structure is required. Thus, a TFRM with the same detector as shown in Figure 5, but with a different refinement structure and different artifact mapping, must be defined to enable the automated refinement depending on the different artifact attached to the QC App node template. The CSARs and the TFRMs for the introduced case studies are available in the public OpenTOSCA repository [44].

VII. Threads to Validity

The SDK-S modeling style demonstrated a high flexibility and generality: By only exchanging the quantum compute node, the execution environment for the quantum circuit to be executed can be changed. Also the integration with non-quantum applications has been demonstrated, even though the integration logic is encapsulated in the API DA that contains the overall invocation, deployment, and execution logic. To facilitate that the API DA is universal for all quantum application using a particular SDK, the implementation of the quantum algorithm must follow a particular programming model to ensure that in can be called by the API DA. This is currently not automated, but existing approach as presented by Zimmermann et al. [45] to generate method stubs based on defined application interfaces can be integrated.

VIII. Related Work

So far, there are still few works dealing with the automated deployment and orchestration of quantum applications. Sim et al. [46] presented the algo2qpu framework that supports the selection of quantum algorithms, their compilation, and execution on a quantum cloud service. They focus on supporting the development process of quantum algorithm rather then the integration with non-quantum applications. We also use existing technologies for deployment automation and focus on the best possible support of different stakeholders by provided different modeling concepts with TOSCA4QC. Zapata Computing who was also involved in the development of algo2qpu, provides Orquestra, a workflow-like approach for the design and execution of quantum algorithms [47]. This is an imperative modeling approach, i.e., the individual tasks to achieve the defined goal have to be specified. In contrast, we provide a declarative approach, i.e., the desired state is defined by a topology template and the required tasks to achieve this state are derived by the orchestration engine.

Dreher and Ramasami [48] demonstrated how Qiskit can be packaged as Docker container for executing an algorithm on IBMQ. Such artifacts can be used by our approach as deployment artifact for the Qiskit container. Depending on whether Qiskit is running on a VM or in a Docker environment a respective DA can be selected. This is one advantage of TOSCA that any type of artifact can be packaged and processed. Other technologies, e.g. jupyter notebooks are also often use to enable the development and direct execution of quantum algorithms [49]. From a TOSCA perspective, such jupyter notebooks are deployment artifacts as well.

There are several model transformation approaches [50] and especially for TOSCA several approaches to reduce model complexity by substituting node types with complete topology templates exist [7]. However, since not only a node type is substituted but also the general structure has to be changed in our approach, i. e., the hostedOn relation to the quantum compute node type is changed to a connectsTo relation, a more advanced TOSCA refinement process had to be applied. Thus, we selected the pattern-based refinement approach introduced in previous works [39]–[41] and extended the mapping features. The advanced refinement approach is not limited to quantum applications and can also be used in general topology or pattern-based refinement processes [40].

IX. Conclusion and Future Work

In this paper we presented TOSCA4QC that introduces two deployment modeling styles to automate the deployment and orchestration of quantum applications: (i) the SDK-specific (SDK-S) style, which covers all technical details of a quantum application development, and (ii) the SDK-agnostic (SDK-A) style, which follows the common modeling approach of classical applications, but hides technical details and is therefore not executable. To make it executable it first has to be transformed into a SDK-S model. For this, we introduced a topology refinement approach that enables the transformation of SDK-A models to SDK-S models in an automated manner. We demonstrated with three use cases that the SDK-S modeling style provides the required flexibility and generality for different SDKs and quantum compute resources. Thus, the deployment of a quantum application can be represented either as an executable model, for which no transformation is required, but which is complex, or as a model that follows known modeling approaches, but which does not follow the deployment principles and therefore has to be transformed before it can be executed. As soon as it is possible to host and invoke a quantum circuit on a quantum service the deployment principles hold for both modeling styles.

As we have concentrated on vertical integration in this work, we will continue to focus on the horizontal integration of quantum and non-quantum applications in the future to improve the orchestration capabilities of quantum applications. Also the selection of a quantum cloud service for the execution of a particular quantum algorithm will be further investigated.

Acknowledgment

The authors would like to thank Felix Burk and Leon Kiefer for there valuable input. The work was partially founded by the DFG projects SustainLife (379522012) and ReSUS (425911815) and the BMWi project *PlanQK* (01MK20005N).

References

[1] E. National Academies of Sciences, Medicine *et al.*, *Quantum computing: progress and prospects*. National Academies Press, 2019.

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

[3] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," vol. 574, pp. 505–510, 2019.

[4] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley, Jul. 2010.

[5] M. Wurster *et al.*, "The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies," *SICS Software-Intensive Cyber-Physical Systems*, Aug. 2019.

[6] C. Endres *et al.*, "Declarative vs. Imperative: Two Modeling Patterns for the Automated Deployment of Applications," in *Proceedings of the 9th International Conference on Pervasive Patterns and Applications (PATTERNS)*. Xpert Publishing Services, Feb. 2017, pp. 22–27.

[7] J. Bellendorf and Z. Á. Mann, "Specification of cloud topologies and orchestration using tosca: a survey," 2019.

[8] A. C. Franco da Silva *et al.*, "Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments," in *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER)*. SciTePress Digital Library, Jun. 2017, pp. 358–367.

[9] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Towards automated iot application deployment by a cloud-based approach," in *6th International Conference on Service-Oriented Computing and Applications (SOCA)*, Dec. 2013, pp. 61–68.

[10] A. C. F. da Silva *et al.*, "OpenTOSCA for IoT: Automating the Deployment of IoT Applications based on the Mosquitto Message Broker," in *Proceedings of the 6th International Conference on the Internet of Things (IoT)*. ACM, Nov. 2016, Demonstration, pp. 181–182.

[11] V. Antonenko *et al.*, "C2: General purpose cloud platform with nfv life-cycle management," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2017, pp. 353–356.

[12] M. S. de Brito *et al.*, "A service orchestration architecture for fog-enabled infrastructures," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017, pp. 127–132.

[13] B. Selic, "The pragmatics of model-driven development," *IEEE Software*, vol. 20, no. 5, pp. 19–25, 2003.

[14] U. Breitenbücher *et al.*, "The OpenTOSCA Ecosystem - Concepts & Tools," *European Space project on Smart Systems, Big Data, Future Internet*, pp. 112–130, Dec. 2016.

[15] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, 10th ed. USA: Cambridge University Press, 2011.

[16] E. Rieffel and W. Polak, *Quantum Computing: A Gentle Introduction*, 1st ed. The MIT Press, 2011.

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

[18] H. J. Briegel *et al.*, "Measurement-based quantum computation," *Nature Physics*, vol. 5, no. 1, pp. 19–26, 2009.

[19] D. Aharonov *et al.*, "Adiabatic Quantum Computation Is Equivalent to Standard Quantum Computation," *SIAM review*, vol. 50, no. 4, pp. 755–787, 2008.

[20] Microsoft. (2020) Quantum development kit. [Online]. Available: https://www.microsoft.com/en-us/quantum/development-kit

[21] IBM. (2020) Qiskit. [Online]. Available: https://qiskit.org/

[22] Rigetti. (2020) Docs for the forest sdk. [Online]. Available: http://docs.rigetti.com/en/stable/

[23] A. McCaskey. (2019) Xacc. [Online]. Available: http://xacc.readthedocs.io

[24] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, C. Blank, K. McKiernan, and N. Killoran, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[25] Amazon. (2020) Aws braket. [Online]. Available: https://aws.amazon.com/braket

[26] D. Weerasiri *et al.*, "A Taxonomy and Survey of Cloud Resource Orchestration Techniques," *ACM Computer Surveys*, vol. 50, no. 2, 2017.

[27] U. Breitenbücher *et al.*, "Combining Declarative and Imperative Cloud Application Provisioning based on TOSCA," in *International Conference on Cloud Engineering (IC2E 2014)*. IEEE, Mar. 2014, pp. 87–96.

[28] ——, "Vino4TOSCA: A Visual Notation for Application Topologies based on TOSCA," in *On the Move to Meaningful Internet Systems: OTM 2012 (CoopIS 2012)*. Springer, Sep. 2012, pp. 416–424.

[29] F. Leymann and J. Barzen, "The bitter truth about quantum algorithms in the nisq era," *arXiv preprint arXiv:2006.02856*, 2020.

[30] M. Salm *et al.*, "A Roadmap for Automating the Selection of Quantum Computers for Quantum Algorithms," *Communications in Computer and Information Science (CCIS)*, p. to appear, 2020.

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

[32] ——, *TOSCA Simple Profile in YAML Version 1.0*, Organization for the Advancement of Structured Information Standards (OASIS), 2015.

[33] T. Binz *et al.*, "OpenTOSCA – A Runtime for TOSCA-based Cloud Applications," in *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*. Springer, Dec. 2013, Demonstration, pp. 692–695.

[34] A. Bergmayr *et al.*, "A Systematic Review of Cloud Modeling Languages," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–38, Feb. 2018.

[35] F. Leymann *et al.*, "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.

[36] F. Leymann, J. Barzen, and M. Falkenthal, "Towards a Platform for Sharing Quantum Software," in *Proceedings of the 13th Advanced Summer School on Service Oriented Computing (2019)*, ser. IBM Technical Report (RC25685). IBM Research Division, Sep. 2019, pp. 70–74.

[37] C. Linnhoff-Popien, "PlanQK — Quantum Computing Meets Artificial Intelligence," *Digitale Welt*, vol. 4, pp. 28–35, 2020.

[38] 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.

[39] L. Harzenetter *et al.*, "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.

[40] J. Guth and F. Leymann, "Pattern-based rewrite and refinement of architectures using graph theory," *Software-Intensive Cyber-Physical Systems (SICS)*, pp. 1–12, Aug. 2019.

[41] L. Harzenetter *et al.*, "Pattern-based Deployment Models Revisited: Automated Pattern-driven Deployment Configuration," in *Proccedings of the Twelfth International Conference on Pervasive Patterns and Applications (PATTERNS)*. Xpert Publishing Services, TO APPEAR.

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

[43] University of Stuttgart. (2019) OpenTOSCA Research Prototype. [Online]. Available: https://www.opentosca.org/

[44] ——. (2020) OpenTOSCA Source Code. [Online]. Available: https://github.com/OpenTOSCA

[45] M. Zimmermann, U. Breitenbücher, and F. Leymann, "A TOSCA-based Programming Model for Interacting Components of Automatically Deployed Cloud and IoT Applications," in *Proceedings of the 19th International Conference on Enterprise Information Systems (ICEIS 2017)*. SciTePress, Apr. 2017, pp. 121–131.

[46] S. Sim *et al.*, "A framework for algorithm deployment on cloud-based quantum computers," *arXiv preprint arXiv:1810.10576*, 2018.

[47] Zapata. (2020) Orquestra the unified quantum operating environment. [Online]. Available: https://www.zapatacomputing.com/orquestra/

[48] P. Dreher and M. Ramasami, "Prototype container-based platform for extreme quantum computing algorithm development," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–7.

[49] Qiskit. (2020) Qiskit tutorial. [Online]. Available: https://github.com/Qiskit/qiskit-tutorials

[50] M. Biehl, "Literature Study on Model Transformations," Royal Institute of Technology, Tech. Rep., Jul. 2010.