Towards Quantum-Algorithms-as-a-Service

Manuel De Stefano
madestefano@unisa.it
SeSa Lab - University of Salerno
Fisciano, Italy

Dario Di Nucci
ddinucci@unisa.it
SeSa Lab - University of Salerno
Fisciano, Italy

Fabio Palomba
fplaomba@unisa.it
SeSa Lab - University of Salerno
Fisciano, Italy

Davide Taibi
davide.taibi@oulu.fi
University of Oulu
Tampere University
Tampere, Finland

Andrea De Lucia
adelucia@unisa.it
SeSa Lab - University of Salerno
Fisciano, Italy

ABSTRACT

Quantum computing is an emerging field of high interest. Many companies have started to work on developing more powerful and stable quantum computers. However, developers still struggle to master the art of programming with a quantum computer. One of the major challenges faced is the integration of quantum parts of a system with the classical one. This paper proposes a novel development model called Quantum-Algorithms-as-a-Service (QAaaS). This new model aims to allow developers to abstract the quantum components away from the design of the software they are building. The model leverages Software-as-a-Service and Function-as-a-Service to support multiple quantum cloud providers and run their algorithms regardless of the underlying hardware.

CCS CONCEPTS

• **Computer systems organization** → **Quantum computing**.

KEYWORDS

Quantum Software Engineering; QaaS; Quantum Computing; XaaS

ACM Reference Format:
Manuel De Stefano, Dario Di Nucci, Fabio Palomba, Davide Taibi, and Andrea De Lucia. 2022. Towards Quantum-Algorithms-as-a-Service. In *Proceedings of the 1st International Workshop on Quantum Programming for Software Engineering (QP4SE '22), November 18, 2022, Singapore, Singapore.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3549036.3562056

1 INTRODUCTION

Quantum computing technology is now a reality [11, 14], making the 21st century the *"quantum era"* [19]. This technology introduces new concepts such as *superposition* and *entanglement*. The former refers to quantum objects that may assume different states simultaneously, while the latter refers to quantum objects that may be deeply connected without direct physical interaction. These concepts promise to revolutionize program computation [17], even leading to the so-called *quantum supremacy* [3], when a programmable quantum device will solve problems that no classical computer can solve in feasible time. As a result, several major business players, such as IBM and Google, are yearly investing hundreds of millions of dollars to develop hardware and software solutions to support quantum program execution.[1]

Developing large-scale quantum software seems to be still far from reality. However, a new scientific discipline was born to enable developers to design quantum programs with the same confidence as classical programs. This new discipline is called *quantum software engineering* (QSE) [18] and aims to foster the application of traditional software engineering methods to quantum programming. A key point of QSE is the coexistence of traditional and quantum systems, which build the so-called hybrid systems [18, 24]. However, this does not come without issues; thus, researchers investigate quantum developers' challenges when dealing with these programs.

To this aim, Khan *et al.* [13] conducted a systematic literature review on software architecture for quantum computing systems. They found that the most common quantum software architectural patterns are layered and pipe-and-filter patterns. Nevertheless, these patterns are general-purpose or classic patterns that can be applied to any software system. For this purpose, more research effort is needed to develop specialized architectural patterns that can better exploit quantum characteristics, e.g., *superposition* and *entanglement*. Weigold *et al.* [22] depicted a set of patterns that can be used to encode classical data into quantum states, which was later extended to provide a broad set of techniques [21]. Gill *et al.* [9] depicted a series of issues in running quantum algorithms efficiently and effectively due to software and hardware limitations. More recently, De Stefano *et al.* [5] defined a taxonomy of challenges related to quantum programming by surveying quantum computing practitioners. The challenges they identify are related to different aspects of quantum programming, spanning from learning-related to community-related ones. A particular challenge that emerged from their taxonomy is related to the *"software infrastructure"*. Developers complained about rapidly changing API, a severe vendor lock-in, difficulty integrating classical parts with quantum parts of the system, and other issues related to the execution environment.

QP4SE '22, November 18, 2022, Singapore, Singapore

ACM ISBN 978-1-4503-9458-1/22/11…$15.00
https://doi.org/10.1145/3549036.3562056

[1] Boston Consulting Group report: https://www.bcg.com/publications/2021/building-quantum-advantage.

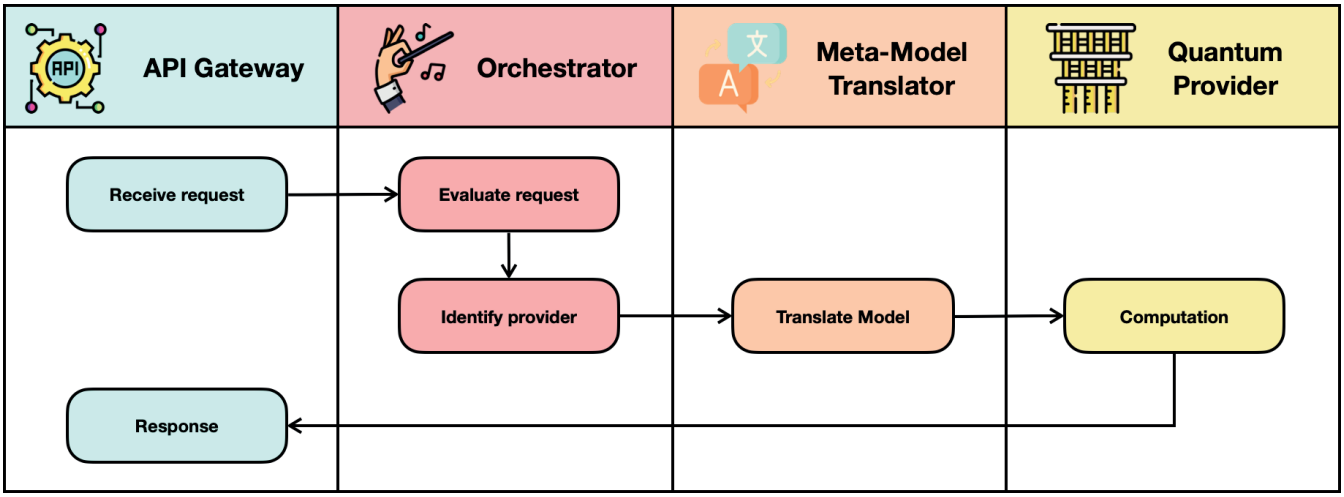

Figure 1: Activity Diagram depicting the high-level process carried out by our model. The API Gateway receives the request, which is forwarded to the Orchestrator. The Orchestrator is in charge of analyzing the desired algorithm and the possible provider to run the algorithm on. Then, the meta-model is translated and sent to the selected provider to run the computation. The result is sent to the Gateway, which fulfills the request.

Researchers have already started proposing possible solutions to overcome the integration issues between traditional and quantum components. Weder *et al.* [20] introduced the Quantum Modeling Extension (QUANTME) to facilitate the representation of quantum circuit invocations in workflows and their orchestration with classical applications. They demonstrated how to make QUANTME workflow models executable on different workflow engines by developing a prototype for three different quantum algorithms and evaluating the achieved reuse and degree of simplification. Zapata Computing [23] developed a commercial solution called Orquestra that is interoperable across all tiers of the stack by running on all major cloud platforms and quantum devices. It separates the structure of a workflow from the details of underlying activities, allowing the user to choose from various techniques for completing a task (quantum or classical).

Kumara *et al.* [15] proposed Quantum Service-Oriented Computing (QSOC). This model-driven methodology enables enterprise DevOps teams to compose, configure, and run enterprise applications without intimate knowledge of the underlying quantum infrastructure. It also advocates knowledge reuse, separation of concerns, resource optimization, and mixed quantum- and conventional QSOC applications.

Moguel et al. [16] proposed a case study highlighting the rough edges and limitations of integrating classical-quantum hybrid systems using service-oriented computing. The conclusion of the study allows us to point out areas where research efforts should be directed to achieve effective quantum service-oriented computing.

Similarly, Gomes *et al.* [10] proposed a collection of ready-made quantum data structures and algorithms to be used by developers. Their work focused on the verification and validation steps by devising techniques to cope with these challenging practices.

Based on this idea, in this paper, we face the integration issues at a higher level of granularity by proposing a novel development model called *Quantum-Algorithms-as-a-Service* (**QAaaS**). This new model aims to allow developers to abstract the quantum components away from the design of the software they are building. It will also allow developers to use the desired quantum algorithms without taking care of the execution environment or the underlying providers by leveraging Software-as-a-Service (SaaS) and Function-as-a-Service (FaaS). Saas is a software licensing and delivery model in which software is licensed on a subscription basis and is centrally hosted. In contrast, FaaS is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure.

The main contributions of this paper are: (i) the proposal of a novel development method for hybrid quantum applications and (ii) a research roadmap to develop and validate this novel methodology.

2 QUANTUM PROVIDERS FACE TO FACE

Many major software companies have started developing and researching quantum computers, which are publicly accessible. Some of them, e.g., Google, IBM, and Microsoft, have already provided their *quantum ecosystem* and released their languages or SDKs; namely, Qiskit [2] (IBM), Cirq [6] (Google), and Q♯ [1] (Microsoft).

All these technologies support the universal gate model of quantum computing to create low-level quantum circuits, compiling them, and executing them on quantum machines [4, 8]. These technologies have different characteristics that could bring specific advantages or disadvantages in terms of syntax, requirements, and computing capabilities. Beyond big popular companies, startups emerged to compete by providing valid alternatives, e.g., Rigetti Computing.[2] Besides the technologies supporting the universal

[2]https://www.rigetti.com/about-rigetti-computing

quantum gate model, some providers support alternative technologies, such as the quantum annealing provided by D-Wave [12]. This model is optimized to combinatorial optimization models, where the search space is discrete with many local minima.

This preliminary evaluation among the available vendors is the first step to defining our proposal, which can bring quantum programming to a higher level of abstraction.

3 TOWARDS QUANTUM-ALGORITHMS-AS-A-SERVICE

Recent studies have shown that developers of quantum applications often struggle with issues related to the software infrastructure, mainly involving the integration between quantum and traditional components. Such issues are caused by several factors, such as the different programming paradigms and languages, the mapping of input between classical and quantum parts, and the very high vendor lock-in that characterizes quantum applications. Furthermore, it has also been shown that quantum frameworks API are in a constant change [5], thus making developers break their code too often.

Based on the idea of Kumara *et al.* [15], this paper presents a development strategy to overcome all these issues coined *Quantum-Algorithms-as-a-Service* (**QAaaS**), recalling the *"Everything as a service"* models that are becoming more and more widespread in cloud development [7]. This strategy will allow developers to write their quantum programs and integrate them into their systems without worrying about the technical details of the quantum providers' platforms. Figure 1 depicts the high-level characteristics of the proposed model in the form of an activity diagram. It consists of four main components, i.e., the API Gateway, the orchestrator, the meta-model translator, and the quantum provider. The API Gateway is a classical API facade that will be accessible through traditional API technologies, like REST, GraphQL, or RPC. External components can interact with the API Gateway to execute quantum algorithms, which will be hosted on the system in the form of a meta-model, i.e., a framework-agnostic model that describes the quantum algorithm to be executed. Then, based on the specific characteristics of the meta-model and the capabilities of the providers to be executed, the orchestrator will forward the request to a specific quantum provider. To this aim, it will first interact with the meta-model translator to obtain a representation of the algorithm compatible with the selected provider. Then, the translated algorithm will be passed to the quantum provider in charge of the execution. Once completed the computation, the resulting output will be returned to the API Gateway that will fulfill the request.

The selection of the best suitable platform does not come without issues. For instance, IBM provides a mechanism to select the least busy machine. However, this cannot be the sole selection criteria. To be executed, the circuit must be transpiled to match the topology and the supported gate set of the selected machine, which requires converting unsupported gates with a series of equivalent gates or adding circuit parts to enable the communication among qubits that are not physically connected. Therefore, circuits will grow in width (i.e., the number of qubits) or depth (i.e., the maximum number of gates on a single qubit), causing performance issues. Choosing the machines whose transpilation requires the minimum alteration of the original circuit could be an available strategy.

Even costs should be considered as selection criteria. Traditionally, in a FaaS context, running a function for a long time costs more. Similarly, in an IaaS context, the best-equipped machine costs more. Thus, choosing the best trade-off between execution time and costs can be another selection criterion.

4 ROADMAP

This section presents a roadmap to guide our future research toward designing and implementing our proposed model.

As a first step, we aim to survey the quantum providers to collect more information about using APIs and frameworks. This step will be necessary to understand how to develop the input mapping strategies. For instance, we already know some standard differences among the endianness of the qubits or the encoding of the rotation angles on some gates: Qiskit puts the least significant qubit in position zero, while other vendors use other standards. This step is necessary to create an encoding strategy to be applied whenever each provider is selected.

After defining a taxonomy of the available quantum vendors with their characteristics and capabilities, the next step will focus on the meta-model translator component. First, we plan to investigate the adaption of currently available open-source solutions. We have already explored some feasible alternatives, such as *Quantum Programming Studio* (QPS).[3] This open-source project has received funding from Rigetti Computing and other private sources. It provides users with a web-based platform to build quantum algorithms and retrieve results by simulating them directly in the browser or running them on real quantum computers. The circuits can be exported into various quantum programming languages/frameworks and run on various quantum simulators and computers. To this extent, the core of this open-source application could be forked and integrated into the proposed model, adapting the code to accept API calls instead of interacting with a web GUI.

Applications in Rigetti are written in JavaScript, while most quantum libraries are developed in Python. Therefore, we need to understand whether re-using the code of Quantum Programming studio and making it interact with the Python code in alternative ways other than function calls or translating it into Python. Suppose we do not find a valid candidate; in that case, we will design and develop a solution to translate any input meta-model to allow for its execution on all the selected quantum providers' platforms. We will adopt already available frameworks to implement the API Gateway. In this case, since the vendors mainly provide libraries in Python, we could develop our API Gateway in the same language, and thus we can rely on libraries e.g., Flask or Django.

Finally, once defined how to design and develop each component, we plan to provide a prototype and validate it using a case study. This study will assess whether the components are correctly developed and whether the quantum algorithms are correctly executed. Then, other non-functional requirements will be explored, e.g., response time, security, and reliability.

Achieving this research agenda will require facing some other issues to be solved, which broadly affect all the proposed framework.

[3] Quantum Programming Studio: https://quantum-circuit.com/

For instance, as pointed out by De Stefano *et al.* [5], changing APIs and vendor lock-in are challenges that affect all quantum applications. To some extent, the proposed framework aims to solve both end-user problems, which can run their quantum code regardless of vendors and specific technologies. The problem will be shifted to a higher level of abstraction where developers will avoid strict vendor specifications and cope with a much more abstract and high-level interface that will be easier to manage.

Testing is an open challenges in quantum programming [5, 24]; therefore, developing such a framework will require testing mechanisms. On the one hand, since the framework will provide API endpoints to the end users, it will be possible to send error codes to verify whether the executions are successful. On the other hand, we will adopt classical testing practices to test the integration among the various platforms.

5 CONCLUSION

This article presents a novel development model coined *Quantum-Algorithms-as-a-Service* (**QAaaS**) aiming to overcome the integration issues of traditional and quantum components. The proposed model provides quantum algorithms as services so developers can access them via traditional API invocations without knowing technical implementation details. We also include a research roadmap to allow the design and implementation of a working prototype and its evaluation.

The realization of this research agenda will represent only a first step towards abstracting quantum computing out of the limitations imposed by hardware providers. In the future, it will be possible to realize a library to facilitate the usage of quantum computing technologies.

ACKNOWLEDGEMENT

Fabio is supported by the Swiss National Science Foundation through the SNF Project No. PZ00P2 186090 (TED). This work has been also partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY).

REFERENCES

[1] 2021. Q#: A Quantum Programming Language. https://qsharp.community. Accessed: 2021-09-21.
[2] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, et al. 2019. Qiskit: An open-source framework for quantum computing. *Accessed on: Mar* 16 (2019).
[3] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature* 574, 7779 (2019), 505–510.
[4] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. *Physical review A* 52, 5 (1995), 3457.
[5] Manuel De Stefano, Fabiano Pecorelli, Dario Di Nucci, Fabio Palomba, and Andrea De Lucia. 2022. Software Engineering for Quantum Programming: How Far Are We? https://doi.org/10.48550/ARXIV.2203.16969
[6] Cirq Developers. 2021. *Cirq*. https://doi.org/10.5281/zenodo.4750446 See full list of authors on Github: https://github .com/quantumlib/Cirq/graphs/contributors.
[7] Yucong Duan, Guohua Fu, Nianjun Zhou, Xiaobing Sun, Nanjangud C Narendra, and Bo Hu. 2015. Everything as a service (XaaS) on the cloud: origins, current and future trends. In *2015 IEEE 8th International Conference on Cloud Computing*. IEEE, 621–628.
[8] Richard P Feynman. 2017. Quantum mechanical computers. *Between Quantum and Cosmos* (2017), 523–548.
[9] Sukhpal Singh Gill, Adarsh Kumar, Harvinder Singh, Manmeet Singh, Kamalpreet Kaur, Muhammad Usman, and Rajkumar Buyya. 2022. Quantum computing: A taxonomy, systematic review and future directions. *Software: Practice and Experience* 52, 1 (2022), 66–114.
[10] Cláudio Gomes, Daniel Fortunato, João Paulo Fernandes, and Rui Abreu. 2020. Off-the-shelf Components for Quantum Programming and Testing.. In *Q-SET@ QCE*. 14–19.
[11] Tony Hoare and Robin Milner. 2005. Grand challenges for computing research. *Comput. J.* 48, 1 (2005), 49–52.
[12] Mark W Johnson, Mohammad HS Amin, Suzanne Gildert, Trevor Lanting, Firas Hamze, Neil Dickson, Richard Harris, Andrew J Berkley, Jan Johansson, Paul Bunyk, et al. 2011. Quantum annealing with manufactured spins. *Nature* 473, 7346 (2011), 194–198.
[13] Arif Ali Khan, Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Tommi Mikkonen, and Pekka Abrahamsson. 2022. Software Architecture for Quantum Computing Systems-A Systematic Review. *arXiv preprint arXiv:2202.05505* (2022).
[14] Will Knight. 2018. Serious quantum computers are finally here. What are we going to do with them. *MIT Technology Review. Retrieved on October* 30 (2018), 2018.
[15] Indika Kumara, Willem-Jan Van Den Heuvel, and Damian A Tamburri. 2021. QSOC: Quantum service-oriented computing. In *Symposium and Summer School on Service-Oriented Computing*. Springer, 52–63.
[16] Enrique Moguel, Javier Berrocal, José García-Alonso, and Juan Manuel Murillo. 2020. A Roadmap for Quantum Software Engineering: Applying the Lessons Learned from the Classics.. In *Q-SET@ QCE*. 5–13.
[17] Leonie Mueck. 2017. Quantum software. *Nature* 549, 7671 (2017), 171–171.
[18] Mario Piattini, Guido Peterssen, Ricardo Pérez-Castillo, Jose Luis Hevia, Manuel A Serrano, Guillermo Hernández, Ignacio García Rodríguez de Guzmán, Claudio Andrés Paradela, Macario Polo, Ezequiel Murina, et al. 2020. The Talavera Manifesto for Quantum Software Engineering and Programming.. In *QANSWER*. 1–5.
[19] Mario Piattini, Manuel Serrano, Ricardo Perez-Castillo, Guido Petersen, and Jose Luis Hevia. 2021. Toward a quantum software engineering. *IT Professional* 23, 1 (2021), 62–66.
[20] Benjamin Weder, Uwe Breitenbücher, Frank Leymann, and Karoline Wild. 2020. Integrating quantum computing into workflow modeling and execution. In *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 279–291.
[21] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. 2021. Expanding data encoding patterns for quantum algorithms. In *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 95–101.
[22] Manuela Weigold, Johanna Barzen, Frank Leymann, and Daniel Vietz. 2021. Patterns for Hybrid Quantum Algorithms. In *Symposium and Summer School on Service-Oriented Computing*. Springer, 34–51.
[23] Zapata. 2020. Orquestra. https://www.zapatacomputing.com/orquestra-platform/
[24] Jianjun Zhao. 2020. Quantum software engineering: Landscapes and horizons. *arXiv preprint arXiv:2007.07047* (2020).