Quantum Software as a Service Through a Quantum API Gateway

Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo, *University of Extremadura, 10003 Cáceres, Spain*

As quantum computers mature, the complexity of quantum software increases. As we move from the initial standalone quantum algorithms toward complex solutions combining quantum algorithms with traditional software, new software engineering methods and abstractions are needed. Nowadays, quantum computers are usually offered in the cloud, under a pay-per-use model, leading to the adoption of the service-oriented good practices that dominate the cloud today. However, specific adaptations are needed to reap the benefits of service-oriented computing while dealing with quantum hardware limitations. In this article, we propose the Quantum API Gateway—an adaptation of the API Gateway pattern that takes into account the fact that quantum services cannot be deployed as traditional services. Instead, the Quantum API Gateway recommends the best quantum computer to run a specific quantum service at run time. As proof of concept, we provide an implementation of the Quantum API Gateway for the Amazon Braket platform.

Quantum computing hardware is rapidly evolving. Different companies are introducing new quantum computers, and the number of qubits keeps increasing.[1] At the same time, new quantum algorithms are being discovered, with quantum software being applied to more domains.

Nevertheless, quantum computers are still costly to own and operate. To expedite the adoption of the quantum paradigm and increase the research efforts in this promising domain, most of the existing quantum computers are offered through the cloud via a pay-per-use model.[2]

The irrefutable success of cloud computing over the past years is demonstrated by recent estimations showing that, in the United States alone, cloud computing contributed approximately US$214 billion in value-added to the GDP and 2.15 million jobs in 2017.[3] This computing model gives companies better control over their costs since they do not have to buy, upgrade, and maintain expensive hardware—they only have to pay for the services they use. A similar model is emerging for quantum computing.

However, quantum software engineering is still in a very early stage.[4] This means that the benefits and advantages provided by cloud computing, which are easily exploited by traditional software, are more elusive when quantum software is involved.

At the moment, most quantum software systems are, indeed, hybrid systems in which quantum software parts coexist and cooperate with traditional software ones.[5] A natural way to approach such collaborative coexistence, especially in the cloud, is by following the principles of software engineering in the services domain and service-oriented computing. Conceptually, the invocation of a quantum piece of software is similar to that of a classical service. However, the current limitations of quantum cloud infrastructure do not allow developers to benefit from their software engineering expertise.

In this article, we propose, using a Quantum API Gateway, a software engineering solution, to simplify the development of hybrid systems that include quantum services. This API Gateway provides a way to treat quantum services as traditional services, so developers can easily integrate them into their processes and tools. At the same time, the Quantum API Gateway takes advantage of the current status of quantum cloud infrastructure, which provides unified access to

1089-7801 © 2021 IEEE
Digital Object Identifier 10.1109/MIC.2021.3132688
Date of publication 9 December 2021; date of current version 28 January 2022.

quantum computers from different vendors—to offer some additional benefits to quantum developers. Specifically, since quantum programs cannot be permanently deployed in a quantum computer—instead, they have to be deployed each time they are executed—the Quantum API Gateway offers a heuristic to determine the best quantum computer in which to deploy the quantum software at run time. Interestingly, this complexity is hidden from the service client. As proof of concept, this article provides an implementation of the Quantum API Gateway by using the services offered by the Amazon Braket Quantum platform.

QUANTUM SOFTWARE AS A SERVICE: STATE OF THE ART

As mentioned previously, most of the quantum computers available for researchers and developers are offered through the cloud in a pay-per-use model. Following a naming scheme similar to that used for traditional computing in the cloud, researchers started calling this model Quantum Computing as a Service (QCaaS).[6]

Given the current status of quantum computers, this access can be best compared with the traditional Infrastructure as a Service, where users directly interact with the physical resources in the cloud. Users can develop and execute quantum programs using a quantum programming language supported by the hardware they employ. The execution process involves sending the program to a scheduler that will, in time, send the code to the quantum computer whenever there is a timeslot available.

This has been the starting point for researchers to try to increase the abstraction level of quantum services. However, the early stage of quantum hardware and quantum software engineering techniques for quantum programs has to be considered. At the moment, rapid progress is being made in most areas of quantum software engineering, although most efforts have revolved around quantum programming and quantum programming languages. Additional efforts are still needed to consolidate the methods, processes, and techniques that will lead to quality, reusable quantum software.[7]

In the specific domain of quantum services, the Quantum Application as a Service[8] proposal is of particular interest. Its authors recommend wrapping all quantum application and deployment logic in a traditional application programming interface (API). This API can then take care of input data encoding and launching the quantum software in a QCaaS offering, thus simplifying the integration of the quantum program with traditional applications.

Similarly, Valencia *et al.*[9] proposed wrapping a quantum program in a classical service. When this service is called, the quantum software is deployed in a QCaaS offering at run time. This process allows developers to integrate their quantum algorithms into service-oriented solutions. This same idea was explored further in Rojo *et al.*'s work.[10] This work suggested the creation of quantum microservices with different endpoints for different QCaaS solutions, thus allowing developers to create complex hybrid solutions in which classical and quantum microservices coexisted.

All these proposals, however, focused on quantum software as traditional services, and thus suffered from a similar drawback. Traditional services can be deployed once in a server in the cloud and then called many times by different clients. However, as far as the authors know, currently there is no quantum computer in which this behavior could be replicated. Every time a quantum piece of code has to be executed, it needs to be deployed first. This increases the computational cost of running the software and disrupts the usual workflow of service-oriented solutions.

Some proposals are starting to emerge to address this issue. Wild *et al.*[11] proposed an extension to TOSCA, a well-known standard for the deployment and orchestration of cloud applications. This extension allows developers to create a deployment model that contains all the information needed to automate the deployment of quantum software in coordination with traditional services.

These proposals illustrate the interest in bringing the benefits of service-oriented computing to quantum software. However, the current state of quantum hardware and quantum software engineering does not support the abstraction level required to work with quantum software as services. Indeed, additional research efforts are necessary to bring both worlds together. In this work, we propose an adaptation of one of the better-known patterns in service-oriented computing, the API gateway. Our implementation brings some of the benefits of services to quantum software while also taking advantage of the current state of quantum hardware to optimize the deployment of quantum services.

QUANTUM API GATEWAY

An API Gateway is a service composition pattern developed to allow creating end-user applications based on the composition of different microservices. The API Gateway serves as the system's entry point, routing requests to the appropriate microservices. It can also invoke aggregate results, transform protocols, and implement shared logic.[12]

SIDEBAR: COMMERCIAL QUANTUM CLOUD PLATFORMS

From a commercial point of view, several companies involved in the development of quantum hardware are also interested in its potential as a cloud computing offering. Most of the biggest computing companies in the world are already offering cloud-based quantum computing solutions. IBM's Quantum Services,[1] Google's Quantum Computing Service,[2] or Microsoft's Azure Quantum[3] are the prime examples. These companies offer their quantum hardware and services through their own platforms—most of the time alongside access to quantum hardware and software from other vendors, unified through their own quantum development ecosystem.

Amazon follows a slightly different approach. As the market leader on cloud computing through AWS, Amazon also offers its own quantum cloud, Amazon Braket.[4] However, Amazon is not developing its own quantum hardware. Instead, Braket focuses on providing services over third-party quantum hardware.

A different approach is also followed by smaller quantum hardware companies like D-Wave.[5] These companies are developing their own quantum computers and offering their services through their own cloud platforms. However, at the same time, they are also integrated in the bigger companies' cloud platforms, in order to increase their reach.

REFERENCES

1. https://www.ibm.com/quantum-computing/services/
2. https://quantumai.google/quantum-computing-service
3. https://azure.microsoft.com/en-us/services/quantum/
4. https://aws.amazon.com/en/braket/
5. https://www.dwavesys.com/

Challenges and Problems

In this article, we propose an adaptation of the API Gateway to address some of the current problems in quantum services. Namely, since quantum services cannot be deployed on a quantum computer to be invoked several times, the Quantum API Gateway would be in charge of deploying the quantum service whenever a client requests it. Considering that this deployment incurs costs with every invocation, the Quantum API Gateway optimizes the deployment strategy. Whenever a quantum service is called, the Quantum API Gateway will decide at run time which of the available quantum computers is best suited for that particular execution, thus optimizing the quantum service invocation process.

To make this decision, the Quantum API Gateway will not only use any information available that is useful in this regard like quantum computers availability, economic cost of the quantum hardware usage, estimated response time, etc., but also current traits of quantum computers like qubit topology, error rate, or fidelity. The availability of this information is dependent on the quantum platform supporting the Quantum API Gateway and, currently, very limited. As quantum platforms mature, more of this kind of information will be provided to users, and therefore, the capabilities of the Quantum API Gateway will be increased.

Calling a Quantum Service

Figure 1 exemplifies the process of calling a quantum service through the proposed Quantum API Gateway, following the steps detailed in the following.

The process starts when a client needs to call one of the quantum services available through the Quantum API Gateway. As shown in step 1, the client needs to specify the service they are invoking, the input parameters of the service (if necessary), and the execution optimization parameters.

Considering the optimization parameters provided, the Quantum API Gateway determines which of the quantum computers available is best suited to attend to the current request. For this purpose, the quantum computer recommender requests all the information available to the QCaaS Provider (step 2), and this one sends it the updated status information of the available quantum hardware (step 3).

Once the best quantum computer to run a given service invocation is determined, the quantum service manager requests all the necessary information (step 4) to deploy the service to the selected quantum hardware with the appropriate input parameters (step 5).

Finally, once the service has finished its execution, the Quantum API Gateway receives the response offered by the selected quantum computers (step 6). Only then it sends back the results to the service client (step 7).

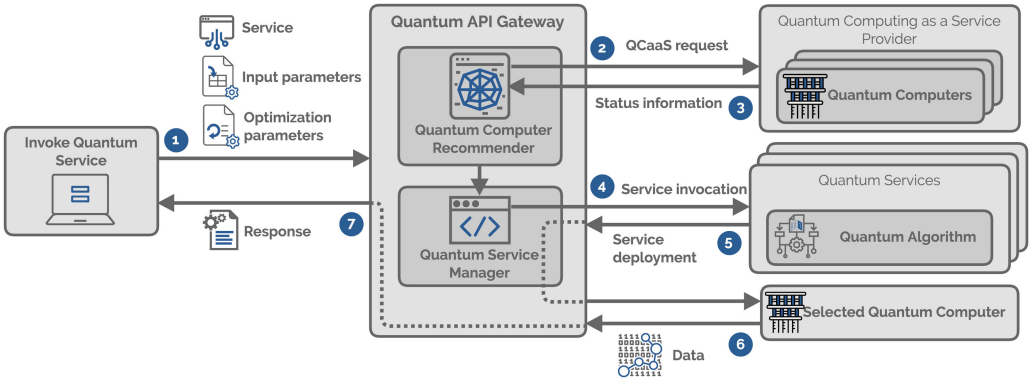

FIGURE 1. Quantum API Gateway—Service calling process.

This process allows choosing the most appropriate quantum computer at run time without significantly increasing the computational cost—since the quantum service would have to be deployed for each invocation anyway. To demonstrate the viability of this proposal, the following section presents an implementation of the Quantum API Gateway for the Amazon Braket platform.

QUANTUM API GATEWAY FOR AMAZON BRAKET

This article presents an implementation of the Quantum API Gateway concept for the Amazon Braket quantum computing platform using Python and Flask. This platform was selected because it offers an integral solution for executing quantum code in different quantum processors. The implementation of the API Gateway is available here.[a]

To determine the best quantum computer for each execution, the Quantum API Gateway takes into account parameters such as the type of code to be executed (gate-based or annealing), the number of qubits required to execute the service, the maximum economic cost for the execution (considering the number of shots needed), or the tradeoff between execution time and cost—to determine whether it is better to select the cheaper machine or the one that is estimated to give the results in the shortest possible time. These parameters are compared against a series of static and dynamic information that Braket offers regarding the quantum computers available to determine the best-suited one, following the algorithm detailed in Figure 2. Other traits of current quantum computers, such as qubit topology or error rate, are not offered programmatically by Braket; therefore, these cannot be taken into account. If the platform offers more information in the future, this could be integrated into the selection algorithm.

Insights and Key Contributions

By selecting the quantum computer for each execution, the implementation of the API Gateway provides the following advantages over the existing support for quantum services.

First, depending on the type of code to be executed, the API Gateway allows developers to choose at runtime (assuming that both implementations of the quantum service are available) between gate-based or annealing machines. In the case of gate-based machines, the number of qubits required is used to determine those with enough computing power. The number of qubits per machine is provided by Amazon Braket as static data.

The next step is to evaluate the cost per selected computer. To that end, the number of shots indicated by the developer, the cost per shot, and the desired maximum cost threshold are used—considering "shots" as the number of repetitions required to identify the solution and, hence, the results of the service execution.

In particular, the *economic cost* of the service execution is calculated as *cost per execution+(cost per shot*N shots)*. Those computers with a result below the indicated threshold are selected. The *cost per execution* and the *cost per shot* can be obtained from Amazon Braket.

Once the machines below the cost threshold have been selected, the Quantum API Gateway uses the Amazon Braket API to check the availability of the quantum computers that meet the requirements. This step is the most time-consuming step, since it

[a]https://github.com/frojomar/Quantum_API_Gateway

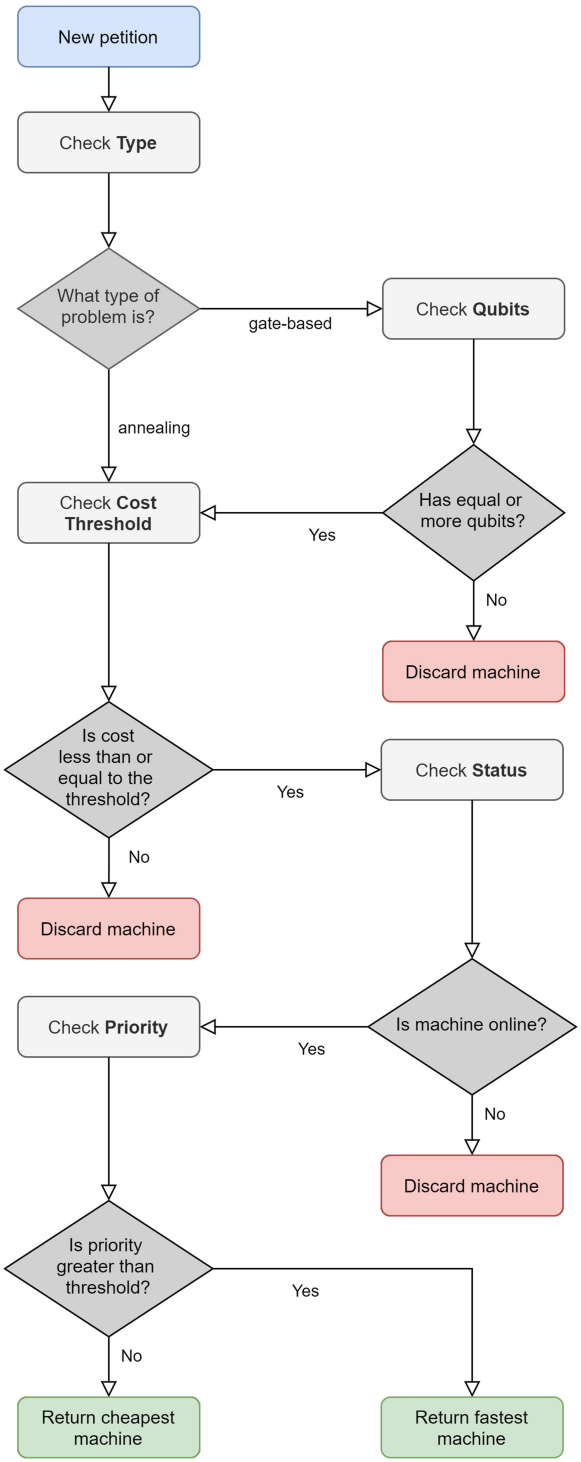

FIGURE 2. Quantum computer selection algorithm for Amazon Braket.

requires communicating with Amazon Braket to establish the actual status of each machine. That is why, it is performed at this point—once the machines meeting the developer's requirements have been selected. Only those machines with "ONLINE" status are selected for the next phase, in which the estimated execution time for each machine is calculated. Then, a tradeoff—*priority* in Figure 2—between the cost and the response time is used to select the machine in which the quantum service will be executed.

Calculation of the estimated execution time for each computer is based on the characteristics of the execution, its context (namely the day of the week and the time of the start of the execution), and the actual time taken by past executions performed on that computer. In addition, another time variable is added—related to the analysis of the previous executions as a time series. Thus, the resulting time contains both the waiting time in the machine's execution queue and the time required to execute the quantum service.

In any case, this implementation of the Quantum API Gateway is modular. Both the static and dynamic characteristics, as well as those taken into account when estimating the execution time, can be complemented with different parameters from other providers. Likewise, the method for analyzing the time estimation information could be replaced or parameterized with different values.

Execution Time Forecasting Model

The model employed to forecast the full execution time—waiting time plus execution time itself—has been developed with Keras. It uses the information available to calculate the execution time, in addition to the context parameters. The provided implementation uses two different models: one for gate-based and one for annealing machines. It has been divided into two models because each one includes specific parameters for each kind of machine.

Moreover, the execution time will depend not only on the characteristics of the specific execution but also on the task load of the quantum processor. For this reason, a temporal analysis is carried out on the last executions performed on each machine. Thus, the characteristics and the time required for each execution are taken into account to predict the execution time. This replaces the temporal analysis of the last minutes of evolution of the processor's workload—since, at the time of writing this article, Amazon Braket did not provide this parameter.

Deep learning techniques, such as neural networks, were used to make these predictions—specifically recurrent neural networks (RNNs),[13] since the

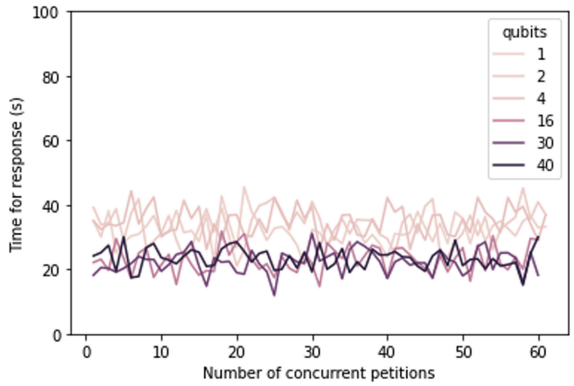

FIGURE 3. Execution time of Quantum API Gateway as a function of the concurrent petition.

analysis took into account a temporal dimension. These neural networks have the ability to maintain data memory. Long short-term memory, in particular, was chosen because its long-term memory is better than that of the classic RNNs. Due to space restrictions, no specific metrics of model quality are provided here. Any other network could be used instead of the proposed one without affecting the Quantum API Gateway behavior—it will only impact the time estimations for the execution of a quantum service.

Main Functionalities

To achieve the abovementioned behavior, the Quantum API Gateway provides two main functionalities implemented as endpoints: one to obtain recommendations on which quantum machine to run the service, and another to provide feedback on the time it took to complete an execution to improve the execution time estimation model. These endpoints are given as follows.

GET /execute: It executes the Quantum API Gateway optimization with the above-described input parameters and returns the best-suited computer to execute the code.

POST /feedback: It allows indicating the Quantum API Gateway the time required to execute the service in a specific computer, alongside the executed service's description—qubits and shots—and context variables—day of the week and time of the day—to improve the execution time estimation model.

Thanks to this, Quantum API Gateway offers developers a tool that enables their users to enjoy a quantum experience more adapted to their needs. When a quantum solution with the lowest possible cost is required, Quantum API Gateway will recommend running on the machine that offers the appropriate features with the lowest possible cost. When fast execution is required, it will recommend the one offering the shortest execution times. In any of these scenarios, the most adequate machine is always identified and provided. In addition, by developing Quantum API Gateway as a REST API, its integration with other service deployment tools is enhanced, improving its integration into current software development tools and methodologies.

VALIDATION

To demonstrate the feasibility of the Quantum API Gateway, we have tested its implementation using the services offered by Amazon Braket platform. The main goal of this validation was to test the delay caused by the inclusion of the Quantum API Gateway in the execution of a quantum service.

To this end, a series of concurrent requests were sent to the gateway, measuring the time it took for the gateway to respond. Specifically, up to 60 concurrent requests were sent. In this way, it was tested whether the time it took for the gateway to respond was stable or worsened as the number of simultaneous requests increased. Furthermore, this process was replicated with different qubit configurations.

Figure 3 shows the results obtained. It is noticeable that the gateway's performance was stable, regardless of the number of concurrent requests. Furthermore, the higher the number of qubits, the less time it took for the gateway to respond, again regardless of the number of concurrent requests. This is due to the decrease in the number of quantum computers that could address the petition for those with a sufficient number of qubits. Therefore, there were fewer computer whose real-time status had to be checked.

As can be seen in Figure 3, a request on the Quantum API Gateway took on average 26.671 s. Of these, 26.668 s (99.98% of the Quantum API Gateway time) were consumed in querying the real-time status of the different machines available through the Amazon Braket API. The execution of the recommendation algorithm took less than 0.01 s. The querying time could be significantly reduced by using a cache mechanism to check the status of the quantum computers—at the cost, however, of losing precision. Another possibility could be the platform providing a real-time API to check this information.

For the sake of brevity, a more detailed description of the validation process and the data obtained, alongside a Python3 Jupyter notebook for replication, can be found at the Quantum API Gateway repository mentioned previously.

CONCLUSION

As the complexity of quantum software increases, so does the need for software engineering methods and tools that enable developers to deal with such complexity. Since most of the current quantum computers are offered through the cloud, it is only natural that software engineers reach for the service-oriented toolbox that has already proved highly successful in the cloud. However, the current limitations of quantum hardware hinder the direct translation of these techniques. Specific adaptations are needed to deal with the particularities of quantum computers.

In this work, we have proposed an adaptation of the API Gateway service-oriented pattern. The Quantum API Gateway addresses the fact that services cannot be deployed on a quantum computer to be executed later. Instead, quantum services have to be deployed before each execution. The proposed Quantum API Gateway deals with this inconvenience by providing an optimization that allows developers to determine the best-suited quantum computer for each execution of a quantum service at run time.

To demonstrate the viability of the Quantum API Gateway, an implementation is provided for the Amazon Braket platform. This implementation takes into account the status of the different quantum computers integrated into the platform, the maximum cost allotted for the execution, and the estimated response time, among other parameters, to recommend the best-suited hardware for each service execution.

The validation performed shows the viability of the Quantum API Gateway, although better integration is needed with quantum computing platforms such as Amazon Braket. As quantum platforms evolve and more information is provided to developers regarding the status of the quantum hardware, the Quantum API Gateway could be adapted to provide better recommendations. At the same time, more complex heuristics could be developed for quantum hardware recommendation—taking into account additional parameters. Similarly, as quantum software evolves, more responsibilities could be placed on the Quantum API Gateway—as is the case with the traditional API Gateway.

ACKNOWLEDGMENTS

This work was supported in part by the projects Interreg V-A España-Portugal 2014-2020 under Grant 0499_4IE_PLUS_4_E and MCIU/AEI/FEDER, UE under Grant RTI2018-094591-B-I00, in part by Grant FPU19/03965, in part by the Department of Economy and Infrastructure of the Government of Extremadura under Grant GR18112 and Grant IB18030, and in part by the European Regional Development Fund.

REFERENCES

1. E. R. MacQuarrie, C. Simon, S. Simmons, and E. Maine, "The emerging commercial landscape of quantum computing," *Nature Rev. Phys.*, vol. 2, no. 11, pp. 596–598, 2020.
2. F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, and K. Wild, "Quantum in the cloud: Application potentials and research opportunities," in *Proc. 10th Int. Conf. Cloud Comput. Serv. Sci.*, May 2020, pp. 9–24.
3. C. Hooton, "Examining the economic contributions of the cloud to the United States economy," Internet Assoc., Washington, DC, USA, Rep., Mar. 2019.
4. M. Piattini, G. Peterssen, and R. Pérez-Castillo, "Quantum computing: A new software engineering golden age," *ACM SIGSOFT Softw. Eng. Notes*, vol. 45, no. 3, pp. 12–14, 2020.
5. A. J. McCaskey, D. I. Lyakh, E. F. Dumitrescu, S. S. Powers, and T. S. Humble, "XACC: A system-level software infrastructure for heterogeneous quantum-classical computing," *Quantum Sci. Technol.*, vol. 5, no. 2, 2020, Art. no. 024002.
6. M. Rahaman and M. M. Islam, "A review on progress and problems of quantum computing as a service (QCAAS) in the perspective of cloud computing," *Glob. J. Comput. Sci. Technol.*, vol. 15, no. 4, pp. 15–18, 2015.
7. J. Zhao, "Quantum software engineering: Landscapes and horizons," 2020, *arXiv:2007.07047*.
8. 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," in *Cloud Computing and Services Science - 10th International Conference, CLOSER 2020, Prague, Czech Republic, May 7-9, 2020, Revised Selected Papers, Ser. Communications in Computer and Information Science*, D. Ferguson, C. Pahl, and M. Helfert, Eds., vol. 1399. Cham, Switzerland: Springer, 2020, pp. 25–58.
9. D. Valencia, J. Garcia-Alonso, J. Rojo, E. Moguel, J. Berrocal, and J. M. Murillo, "Hybrid classical-quantum software services systems: Exploration of the rough edges," in *Quality of Information and Communications Technology - 14th International Conference, QUATIC 2021, Proceedings, Ser. Communications in Computer and Information Science*. Cham, Switzerland: Springer, 2021, pp. 225–238.

10. J. Rojo, D. Valencia, J. Berrocal, E. Moguel, J. García-Alonso, and J. M. M. Rodriguez, "Trials and tribulations of developing hybrid quantum-classical microservices systems," in *2nd Quantum Softw. Eng. Technol. Workshop, QSET'21, CEUR Workshop Proc.*, vol. 3008, 2021, pp. 38–53.
11. 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 *Proc. 24th IEEE Int. Enterprise Distrib. Object Comput. Conf.*, Eindhoven, The Netherlands, 2020, pp. 125–134.
12. D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," in *Proc. CLOSER*, 2018, pp. 221–232.
13. S. Bhavya and A. S. Pillai, "Prediction models in healthcare using deep learning," in *Advances in Intelligent Systems and Computing*, vol. 1182, Cham, Switzerland: Springer, Dec. 2019, pp. 195–204. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-49345-5_21

JOSE GARCIA-ALONSO is currently an Associate Professor with the Department of Informatics and Telematics System Engineering, University of Extremadura, Badajoz, Spain, and a co-founder of the startups Gloin and Viable. His research interests include quantum software engineering, eHealthCare, eldercare, mobile computing, context-awareness, and pervasive systems. He received the Ph.D. degree from the University of Extremadura (with European Mention) in 2014. He is the corresponding author of this article. Contact him at jgaralo@unex.es.

JAVIER ROJO is currently a Ph.D. student with the University of Extremadura, Badajoz, Spain, where he has a predoctoral position through a grant for the formation of university professors from the Spanish Ministry of Science, Innovation and Universities. His research interests include software engineering, eHealth, pervasive computing, and mobile computing. He received the M.Sc. degree in computer science from the University of Extremadura in 2021. He is a member of the IEEE. Contact him at javirojo@unex.es.

DAVID VALENCIA is a currently a postdoctoral researcher with the University of Extremadura, Badajoz, Spain. His research interests include quantum computing, parallel and distributed computing, and software engineering. He received the M.Sc. and Ph.D. degrees in computer science from the University of Extremadura in 2004 and 2010, respectively. Contact him at davaleco@unex.es.

ENRIQUE MOGUEL is currently an Assistant Professor at the University of Extremadura Badajoz, Spain. His research interests include web engineering, smart systems, and quantum computing. He received the M.Sc. degree from University Carlos III, Getafe, Spain, in 2010 and the Ph.D. degree from the University of Extremadura in 2018, both in computer science. Contact him at enrique@unex.es.

JAVIER BERROCAL is currently an Associate Professor with the Department of Informatics and Telematics System Engineering, University of Extremadura, Badajoz, Spain, and a co-founder of the startup Gloin. His research interests include mobile computing, context awareness, pervasive systems, the Internet of Things, and fog computing. He received the Ph.D. degree from the University of Extremadura (with European Mention) in 2014. Contact him at jberolm@unex.es.

JUAN MANUEL MURILLO is currently a Full Professor of Software Engineering at the University of Extremadura, Badajoz, Spain, and a co-founder of the startups Gloin and Viable. His research interests include quantum software engineering, software architectures, mobile computing, and cloud computing. He received the Ph.D. degree in computer science from the University of Extremadura in 2001. Contact him at juanmamu@unex.es.