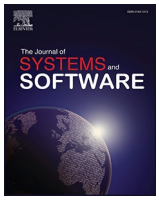
The Journal of Systems and Software 214 (2024) 112061

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



qSOA®: Dynamic integration for hybrid quantum/Classical software systems

José Luis Hevia [a], Guido Peterssen [a], Mario Piattini [b,*]

[a] Quantum Software Technology, Spain
[b] University of Castilla-La Mancha, Spain

ARTICLE INFO

Edited by Prof Raffaela Mirandola

Keywords:
qSOA
SOA
Hybrid systems
Classical/quantum integration

ABSTRACT

This is the "quantum decade" when quantum computers are proving to be more useful for some types of processing than their classical counterparts. The solutions of the future will combine classical IT with quantum algorithms and applications. However, there is very little work that addresses this issue. Each manufacturer provides specific tools for their products, which makes integration between quantum and classical systems a costly and time-consuming process.

We believe that we should follow good software engineering principles and best practices to build platforms and tools that insulate software and systems engineers from implementation details and provide them with adequate support. In this paper we propose qSOA® as a mechanism for the dynamic integration of hybrid (quantum/classical) software systems and present in detail an SDK for Python. qSOA® provides a set of well-defined functions with which a classical system can access quantum products as if they were a piece of classical software, accessing the use case and not its complexities. In addition to some examples that illustrates its operation in real cases, we present the evaluation of qSOA® by 200 quantum software developers in a workshop in collaboration with AWS Braket.

1. Introduction

The current decade (the 2020's) is seen as the "quantum decade", when "quantum computing is poised to expand the scope and complexity of the business problems we can solve" (IBV, 2021), offering a true "quantum advantage". According to the Institute for Business Value, we will witness "the most important computing revolution in the last sixty years" because of the integration of classical computing, quantum computing and artificial intelligence (IBV, 2021).

In fact, completely new solutions are possible in multiple business areas: economics and financial services, chemistry, medicine and healthcare, supply chain and logistics, energy, and agriculture, etc. (GAO, 2021). That is why quantum computing is seen as a strategic technology by the world's leading economies (WEF, 2022), and explains the $764 million invested only in quantum software from 2020 to 2021. Fig. 5

Quantum computing ushers a new era for software engineering (Piattini et al., 2020a; Barbosa, 2020; Zhao, 2020; Ali et al., 2022,), and we are witnessing research proposing different methodologies, lifecycles, techniques, and tools to build true quantum software engineering (QSE) (Zhao, 2020; Pérez-Castillo et al., 2021; Serrano et al., 2022).

However, for quantum applications to be truly useful and not just PoCs (Proof of Concepts), they must be developed according to Software Engineering best practices (Piattini et al., 2020b) to ensure their quality, and platforms should insulate software and system engineers from implementation details, providing them with adequate support. We also stress the importance of starting to develop new techniques and/or adapting classical software engineering techniques to the characteristics of quantum programming. To do so, we will have to, on the one hand, be agnostic with respect to quantum computing technology; and on the other hand, consider the design of hybrid systems that occur in the reality of organisations (Piattini et al., 2021). In fact, to be of value to enterprises, quantum applications or components need to be easily integrated with traditional information systems, where the data necessary for quantum computations reside. The Software Engineering Institute in its "Agenda for Research and Development of Software Engineering" (SEI, 2021) stresses the importance of having different interfaces for quantum software, including application programming ones. For real quantum software adoption is therefore essential to have tools that allow us to build "hybrid" systems by easily and dynamically integrating

* Corresponding author.
E-mail address: mario.piattini@uclm.es (M. Piattini).

https://doi.org/10.1016/j.jss.2024.112061
Received 1 June 2023; Received in revised form 7 April 2024; Accepted 11 April 2024
Available online 23 April 2024

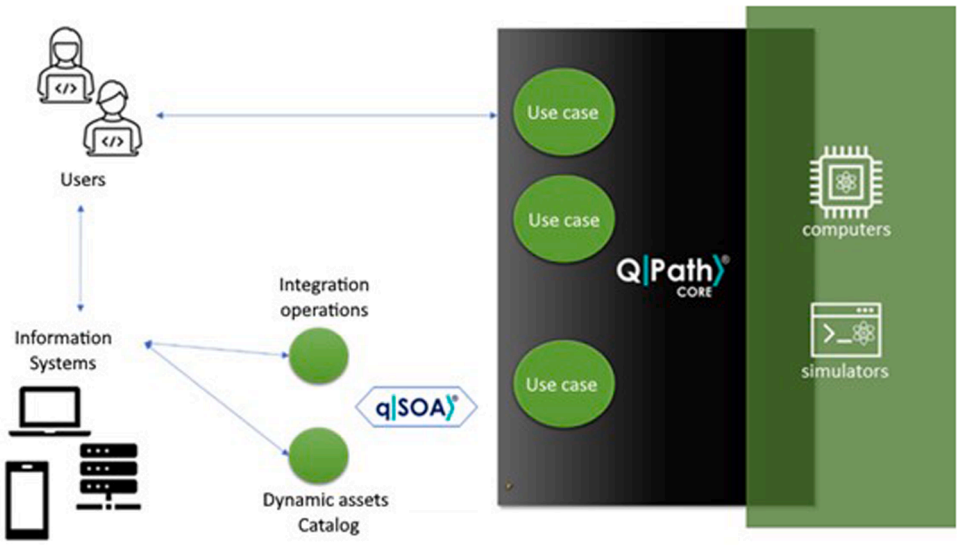


Fig. 1. qSOA® in the QuantumPath® service model.

quantum components with "classical" IT.

Zhao (2020) and Ali et al. (2022) presents a QSE state-of-the-art and landscape, in which it can be seen that no integration proposal exists. Barbosa (2020) addresses the research challenges and indicates the main research directions, including architectural ones, but while existing work addresses the creation of quantum systems, it does not address the integration of classical/quantum systems. Akbar et al. (2023) categorises various QSE challenges, this paper deals with the 14th challenge: "Integration with classical computing", stressing the importance of efficient communication between the classical and quantum components.

Within the framework of the QuantumPath® platform that supports the entire lifecycle of quantum application development from algorithm specification to execution support (Hevia et al., 2021b), we have developed qSOA® as a mechanism for the dynamic integration of hybrid (quantum/classical) systems. Several platforms for quantum software development have been developed, but none of them address the problem of integration between classical and quantum software (Hevia et al., 2021a; Serrano et al., 2023). Moguel et al. (2022) presents an analysis of current quantum software from the point of view of service-oriented computing and uses Amazon Braket to deploy quantum services by wrapping them on a classical service. However, we are not aware of any work that proposes to carry out the integration of the quantum world and the classical world following a system engineering philosophy similar to that of SOA (Service-Oriented Architecture) in the classical world.

The rest of this paper is organized as follows. Section 2 presents a review of the main hybrid (quantum/classical) integration problems. Section 3 summarizes the qSOA® proposal for the dynamic hybrid systems integration. Section 4 presents the QuantumPath qSOA® SDK for Python (PySDK). Section 5 shows three examples of using qSOA®, including one about the integration of a quantum application with an ERP classical system. Section 6 characterizes qSOA®, comparing it with alternative technologies, summarizing some practical implementation challenges, and presenting the results of a survey about the usefulness of qSOA®. Finally, conclusions and future work are included in Section 7.

2. Hybrid systems integration

2.1. State of the technology

It is a fact that quantum computing depends on highly specialized hardware from only a few manufacturers (IBM, Rigetti, Google, d-Wave, etc.). In addition, this hardware has high costs, which do not make the mass distribution of these computers to the market possible, and its use is only possible through channels specifically created by the hardware manufacturer, following RPC access principles and open Internet protocols, i.e., following a model similar to that of the first "classical" mainframes but adapted to the current times. This leads each manufacturer to provide specific tools for their products, which makes the process of learning not only quantum computing technology, but also the elements necessary to exploit it, costly and laborious.

These spectacular machines and their potential services using quantum mechanics will be in the deepest layer of a computing infrastructure (the backend, hosted wherever it is hosted or by a cloud service) until the technology is ready to make it to the market, miniaturised and packaged. And this brings us to how design principles and software product architectures must be designed.

In order to access quantum services centrally, the systems engineering principle does not change, we need to access via a communication network, an access protocol to the services of the new system and best practices to apply at all levels of access (security, traceability, governance…). Such best practices must have already been well established and well proven for a long time.

2.2. qSOA® approach

qSOA® is the result of extensive and in-depth research with the main objective of creating the necessary technologies to be able to bring to the practice of professional software development a method that, as has been successfully done for years in the classical domain, makes available to software system developers a service architecture specially designed to simplify the development of quantum/classical business applications.[1]

Therefore, our proposal of qSOA®, a technology that makes use of open internet standards such as REST API or JSON as message format, is positioned at the service layer of the product and makes it possible to exploit the quantum services platform using a high-level API. This provides a set of well-defined functions running on a reliable, flexible, secure, scalable, and high-performance platform, with which a classical system can access quantum products as if they were just another piece of

[1] qSOA® is not an open-source tool, but anyone who wants to try it out to experience its capabilities can do so using a free developer license, Free Developer, at https://core.quantumpath.app/

Fig. 2. qSOA® API for running quantum business use cases.

classical software.

It is a matter of accessing the use case and not its complexities. Fig. 1. Shows how qSOA® fits in the QuantumPath® service model and its impact on hybrid platform architectures. However, this is not enough since it is necessary to interact with the quantum platform to define new quantum products in a totally dynamic way and adapted in real-time to practical problems, using tools that respond to a distributed model of software components assuring standards, security, encryption, scalability, etc.

By extending the service-oriented architecture to the quantum domain, with qSOA® we propose a clear, efficient, concrete, secure and open-standard architecture for interfacing classical systems to quantum systems, using well-known systems and software, and common tools. A high-level proxy wrapper is provided that enables classical technology to talk to quantum technology, leaving the architectural elements, technical complexities, optimisations, translations, mappings, etc. "inside the box". By simplifying the integration model, we accelerate the inclusion of quantum technology in the classical infrastructure in a natural, unforced way, extending to the quantum domain concepts already tested and consolidated in systems and software engineering.

2.3. qSOA® interaction contexts

With qSOA® we want to provide two interaction contexts:

1) **Access via API to the quantum use cases developed in the system for its exploitation** (controlling its life cycle with the set of assisted graphical tools). The API in this context (Fig. 2) allows access to the taxonomy of the developed assets, facilitating an asynchronous execution in the backend servers of the flows, obtaining the results, and establishing the quantum runtime environment in runtime. This is a very specific model specialized in simplifying access to the front-line use case while always controlling the levels of authorization to the resources depending on the user used to connect.
2) **Automated access to the life cycle of the asset itself, that is, being able to dynamically expand the catalogue of assets** (circuits and dynamic flows, controlling their states, compilations and transpilations in a fully automatic way). This means that the computer systems themselves act as clients, which can dynamically define quantum assets (see Fig. 3) and, for example, overcome design barriers in terms of qubit resolution, or create circuit refactoring based on information taken in the production environment. Once the barrier of the number of qubits supported by current quantum technology has been overcome, the creation of algorithms will itself become exponential.

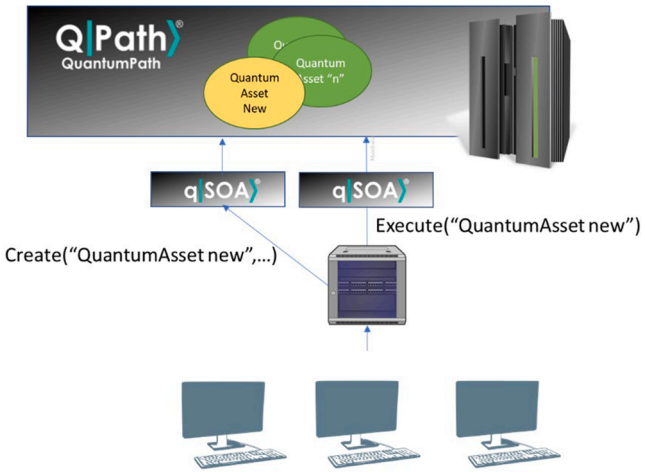


Fig. 3. Dynamic quantum asset using qSOA® API.

Using agnostic technology and metalanguages, developers would "only" need to compose circuits like "DNA strands" through systems. As vendors will change their APIs, capabilities, and requirements (identifying themselves as the lowest level backend), qSOA® will act as a wrapper thanks to its interconnection with QuantumPath® CORE services. If each flow designed corresponds to a "quantum use case", access

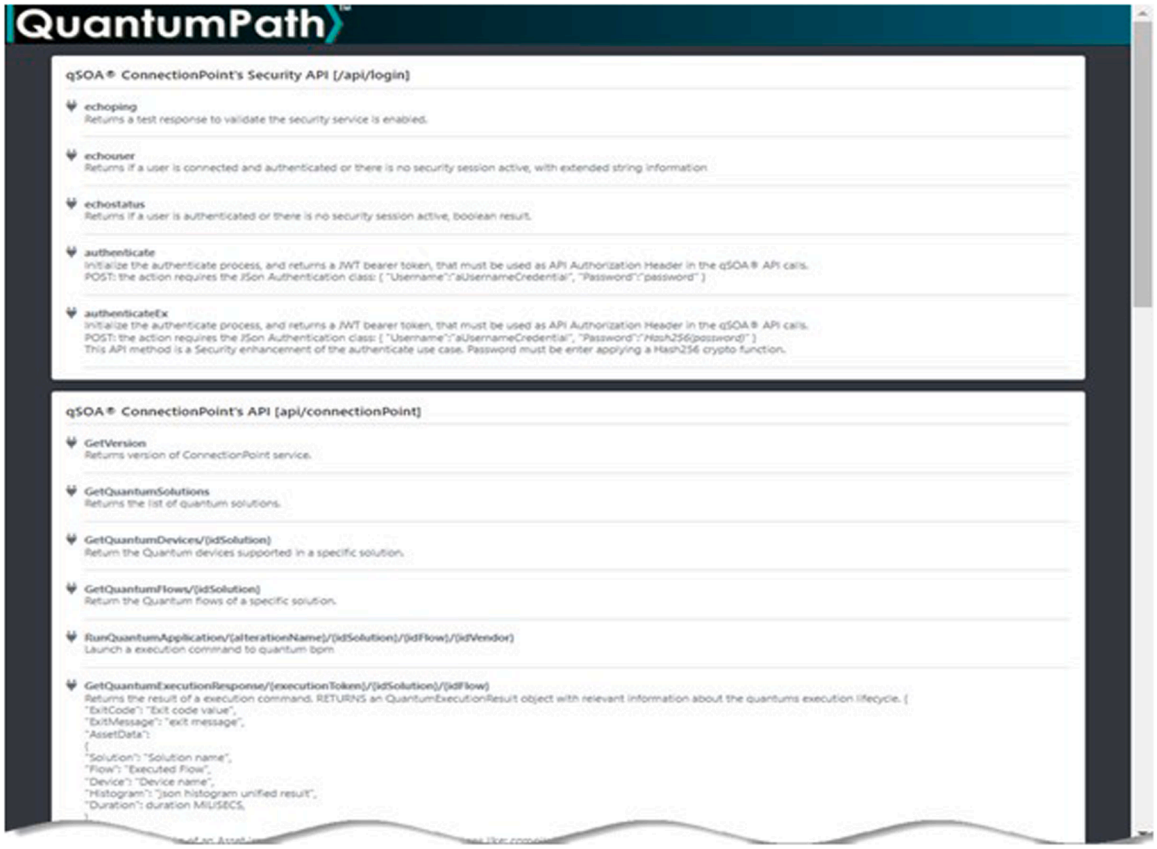


Fig. 4. qSOA® REST API.

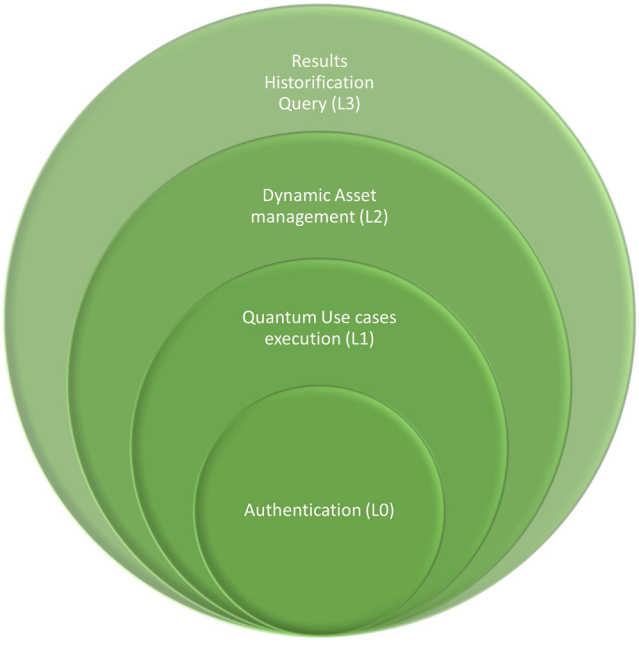


Fig. 5. qSOA® API REST Levels.

from qSOA® to said use case will be possible always in the same way, without being affected by the provider's changes. qSOA® has been conceived, designed, and implemented to be extremely concrete, so it is not an API with hundreds of functions (Fig. 4). On the contrary, it makes the integration process extremely concrete, parameterizable, and controlled. And this has been achieved based on well-tested architectures, designed to be secure, under principles of high performance and scalability that provide tools to development teams that allow them to prepare now to lead the way and be efficient in its evolution. qSOA® makes it possible to access the quantum system with any classical technology (Python, .NET or Java to give examples). Thanks to its implementation based on RPC standards under open protocols, it is more than feasible to build a client with any programming language capable of establishing client sockets, speaking over HTTP protocol and using JSON or XML as message formats.

Therefore, qSOA® provides the framework for the interconnection of a classical system with the quantum system through an open protocol and a high-level software services model that facilitates both the execution of quantum use cases and their lifecycles. In this way, access to quantum technology is transparent to the classical system and, therefore, its integration is simplified, taking advantage of 100 % of the accumulated "classical" software engineering experience.

3. The API of qSOA®

The qSOA® API responds to a web services model based on open standards and protocols facilitating RPC calls to the QuantumPath® CORE. To make the service model very clear, the API connection points—primitives—have been functionally organized into four levels. Each of them is a group of methods with specific purposes for the business rules:

- Authentication and connection
- Execution of quantum use cases
- Dynamic asset extension primitives
- Historic data query for analysis purposes

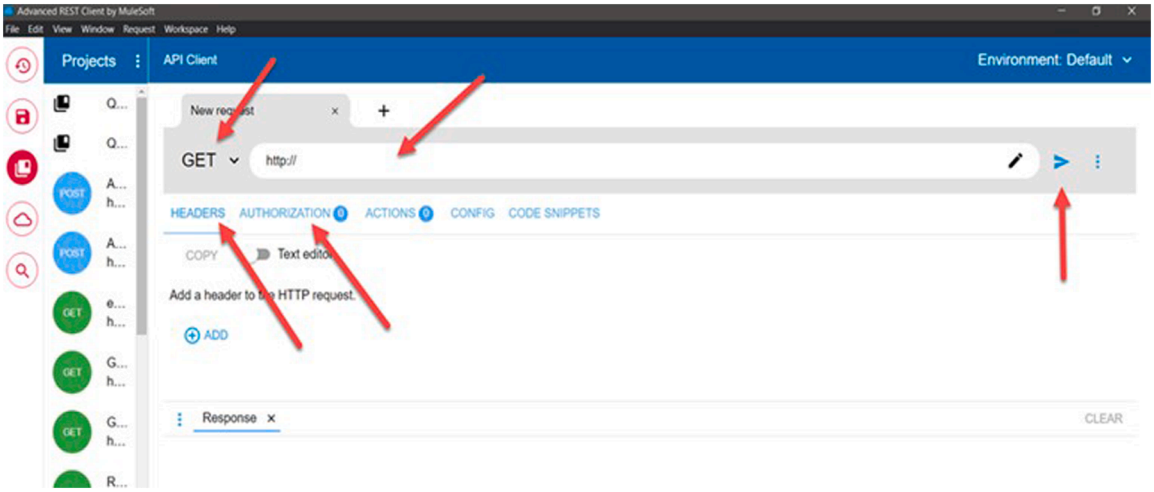


Fig. 6. The ARC environment ready to compose HTTP REST calls.

3.1. Authentication and connection

To enable connections to the qSOA® API at its most basic level, it is necessary to have knowledge and experience in open protocols and to establish connections to distributed systems, using sockets over TCP/IP, the HTTP protocol to establish the dialogue with the server and the JSON format to encapsulate the data structure.

In order to start testing the qSOA® API—without having to write code for it from scratch—we recommend the use of recognized open-source browser extensions/desktop applications that will allow us to interact with the HTTP protocol. This is the case, for example, of *Advanced REST CLIENT -ARC* (https://docs.advancedrestclient.com)- or cURL, to mention the most representative utilities. Of course, it is possible to download the QuantumPath® SDK for developers written in python or .NET to avoid this "low level" exploration…

This type of tool will enable users to set up, step-by-step, the call data and compose the GET/POST bodies necessary to establish the dialogue with the protocol (Fig. 6). Basically, we will have to focus on:

- Establishing the HTTP command that will launch the action to the server, which can be GET (basic calls with parameters in the URL) or POST (advanced calls, with BODY bodies to encode more complex messages).
- Defining the URL of the server to be called and the REST path pointing to the service exposing the primitives.
- Setting the HEADERS that will allow us to "inject" protocol elements that establish the context of the call.
- Completing the BODY that will allow us, in POST calls, to define a formatted message to the server as a parameter required by the established REST action.

As an example, the following is a description of the call: qSOA® ConnectionPoint's Security API [/api/login]

authenticate Initializes the authenticate process, and returns a JWT bearer token, that must be used as API Authorization Header in the qSOA® API calls.

POST: the action requires the Json Authentication class: { "Username":"aUsernameCredential", "Password":"password"}

With the ARC tab empty, we fill in the following data:

HTTP property	Worth
Method	POST
URL	https://qsoa.quantumpath.app:8443/api/login/authenticate
Content-Type	Application/json
BODY	{ "Username": "username", "Password": "password" }

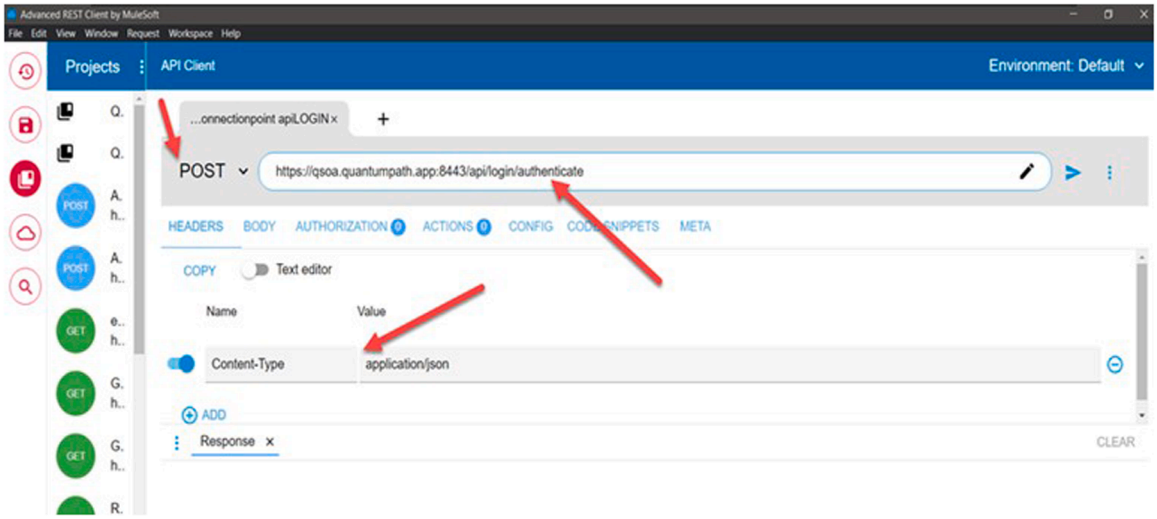


Fig. 7. Prepare the HTTP call properties to authenticate against the qSOA® REST API.

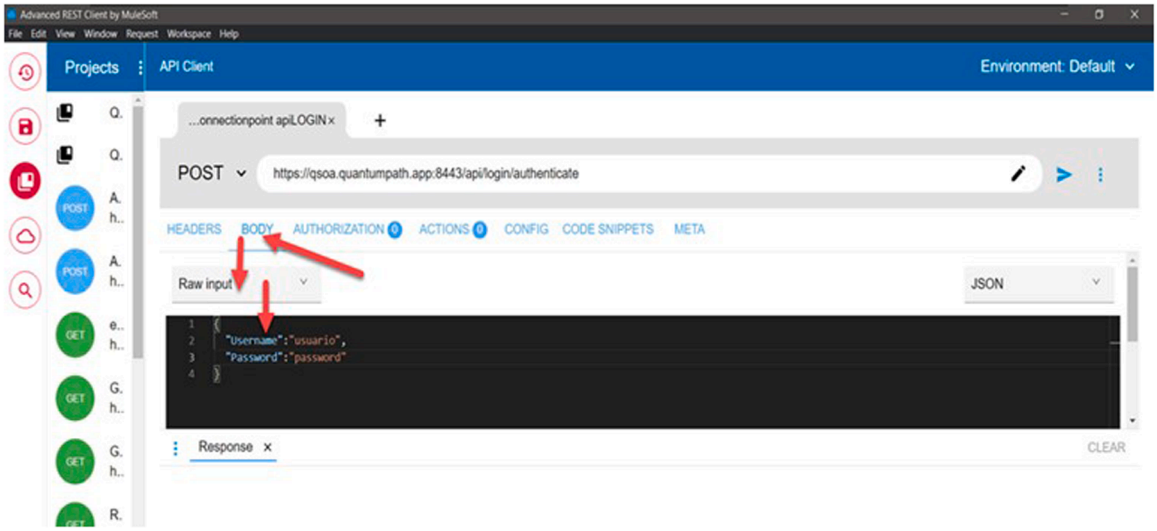


Fig. 8. Prepare the BODY with the call message.

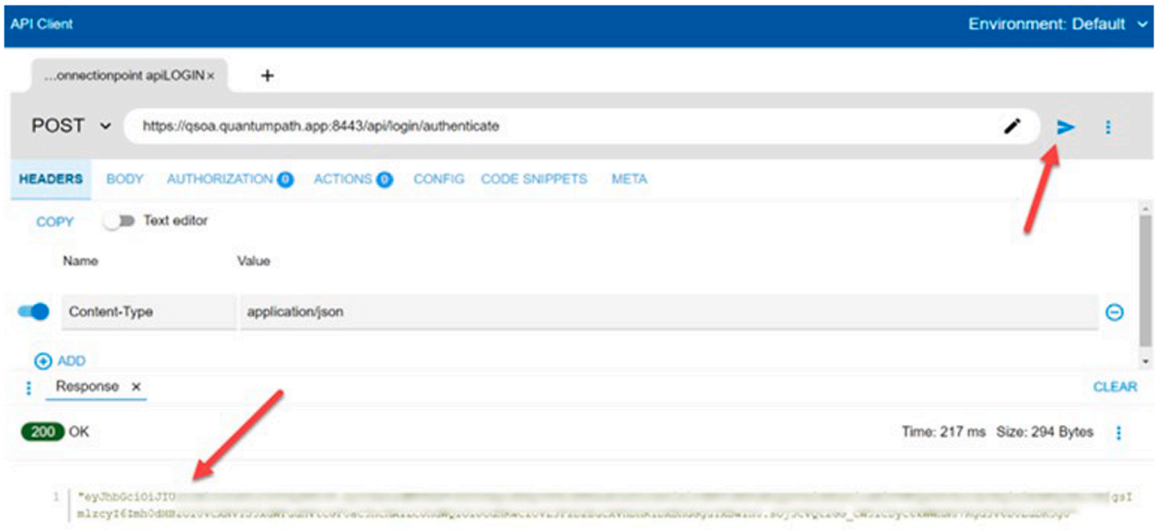


Fig. 9. The response to the authentication primitive.

The ARC visual environment will be as in Figs. 7 and 8.

If all goes well, when making the HTTP call, the server will respond with the *authentication TOKEN* as a response (Fig. 9). This token must be saved to notify it in the rest of the calls that we make to the API.

Of course, if all goes well, the protocol returns the response code 100 "OK". In any other case, the 40x error and the cause of the error. For example, 401 "Unauthorized" if the credentials were not correct.

Once we have obtained the "*authentication token*" or "*bearer TOKEN*", we are now ready to make calls to the REST API to primitives that interact with the QuantumPath® CORE. As the first interaction test, we will collect the version string. For this, we will use the API reference: qSOA® ConnectionPoint's API [api/connectionPoint]

GetVersion Returns version of ConnectionPoint service.

In a new ARC tab, we will now fill in the new data of the qSOA® primitive, using the following as data:

HTTP property	Value
Method	GET
URL	https://qsoa.quantumpath.app:8443/api/connection/GetVersion
Authorization	Bearer <<Obtained Authentication Token>>

In this manner, once the protocol properties have been introduced, the ARC interface remains as shown in Fig. 10, and considers that it has been used to execute the HTTP command. The result is the string "QuantumPath qSOA services -Development context-".

We could now call all the primitives offered in the QuantumPath® REST API. Again, we should mention the simplicity of the API, which offers access to a completely quantum agnostic technology. This means that either by exploiting the quantum applications generated with the graphical tools or by dynamically generating the quantum applications through client applications, there is an unlimited range of possibilities where the classical world can industrially address transparent integration with the "quantum use cases".

Obviously, this level of access to the REST API provides a low level of entry to anyone who wants to generate their access points to the platform using any classic technology that has the possibility of opening a communication channel. To reduce this effort to already known technologies, qSOA® provides higher level SDKs to IT teams. Among the available technologies are those most massively adopted in the industry: NET Framework, Python, and in the future, Java.

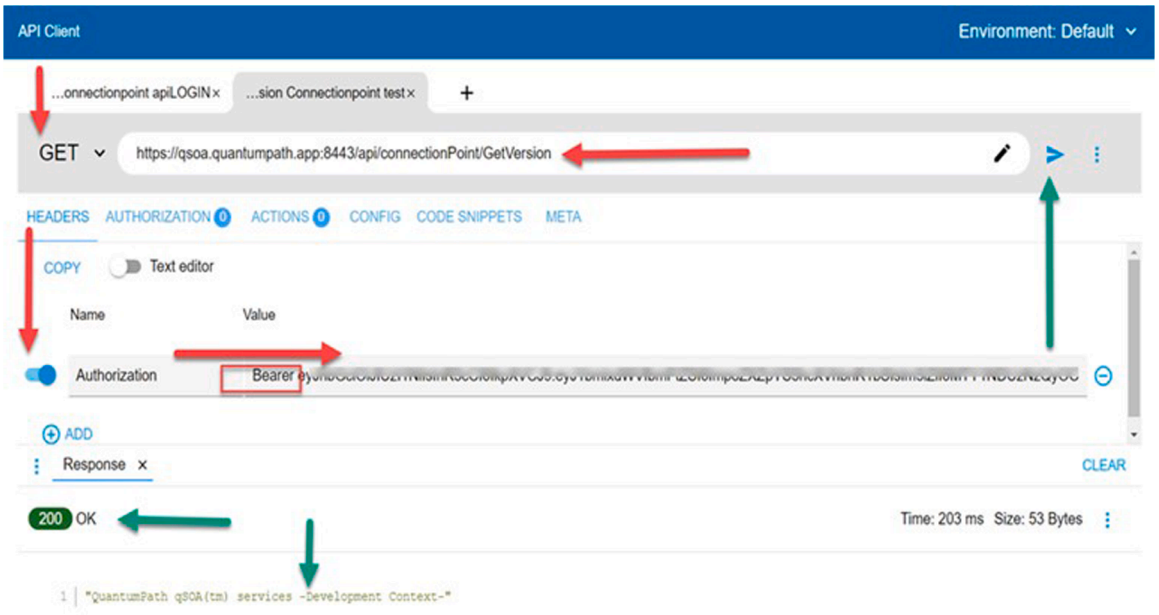


Fig. 10. Running the qSOA® GetVersion primitive.

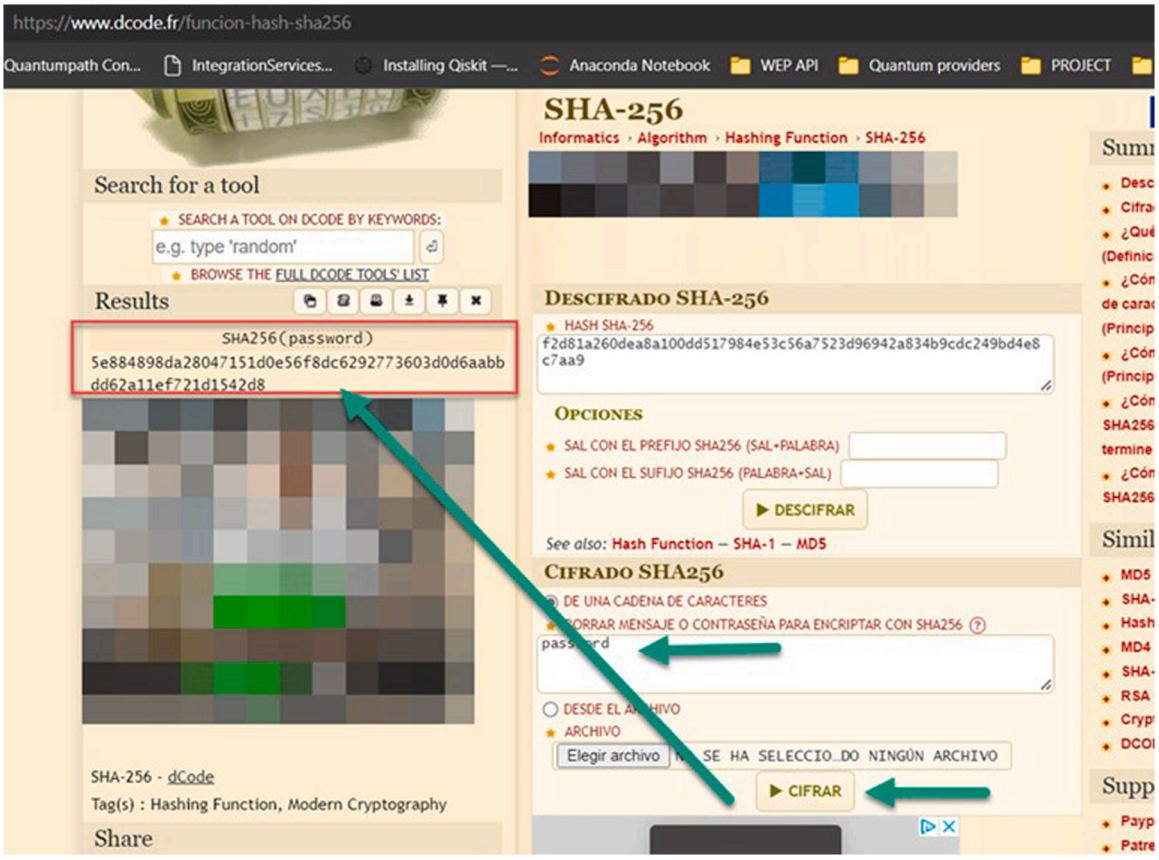


Fig. 11. Obtaining a HASH256 of a word manually.

Before being able to execute primitives, it will always be a first requirement to establish an access negotiation. And this will be done using the "Authenticate" primitives. By means of these primitives, we will be able to pass to QuantumPath® the user credentials with which the system will recognize us and authorize access to the assigned resources.

Primitive	Purpose
Authenticate	Start the authentication process, informing the user and password to access the system
AuthenticateEx	Start the authentication process, informing the user and a 256 hash of the system access password.

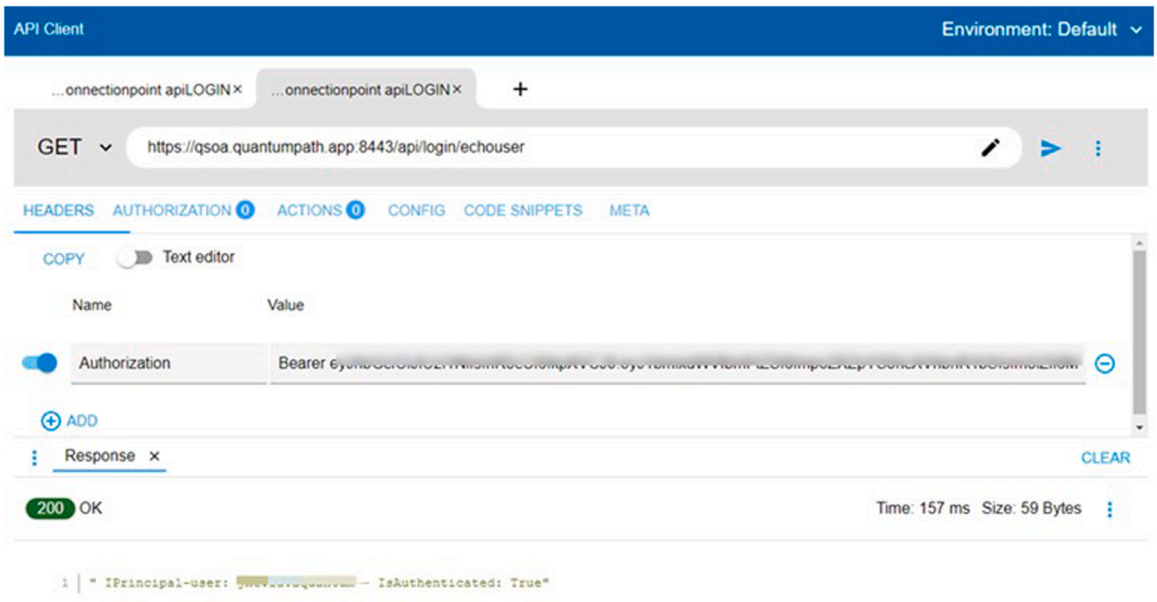


Fig. 12. Example of primitive "Echoser".

Once the authentication token has been obtained, we will be able to access the rest of the primitives, using the "Authorization" HTTP header in all of them and establishing the value of the token obtained in the authentication process.

We will see graphically how this is authenticated with the AuthenticateEx method, which allows us to enter the password encoded in a HASH256 format. This format is more secure since from the same input, the credential data will never be exposed.

In this case, to generate the HASH256 string, for example, with the service https:\\www.dcode.fr\function-hash-sha256 (Fig. 11), we will imagine that we want to use "password" as the password. HASH256 will be: 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8

And the call to the qSOA® API would be:

HTTP property	Value
Method	POST
URL	https://qsoa.quantumpath.app:8443/api/login/authenticate
Content-Type	Application/json

BODY	{
	" Username ": " username ",
	" Password ": " 5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8 "
	}

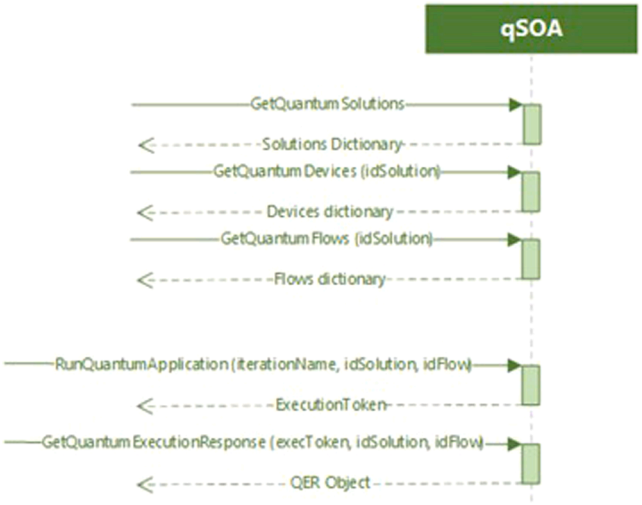


Fig. 13. Sequence diagram for a flow execution.

Fig. 12 shows a practical example of how to validate the connected user; remember that the authentication token must be entered in calls to the API.

3.2. Execution of quantum use cases

The "connection point primitives" encompasses the execution cycle control actions of the quantum assets managed by QuantumPath®. With the platform, the user can graphically work on the complete life cycle of the elements that make up a quantum solution: the solution and its connections, the quantum circuits, and the flows responsible for orchestrating their execution. This means that the entire development process is carried out from QuantumPath® and when the time comes, they are published to be consumed by classic clients. And this is where this group of qSOA® primitives becomes important.

Again, consulting the documentation provided in the qSOA® system, the primitives available to have the necessary information to execute the

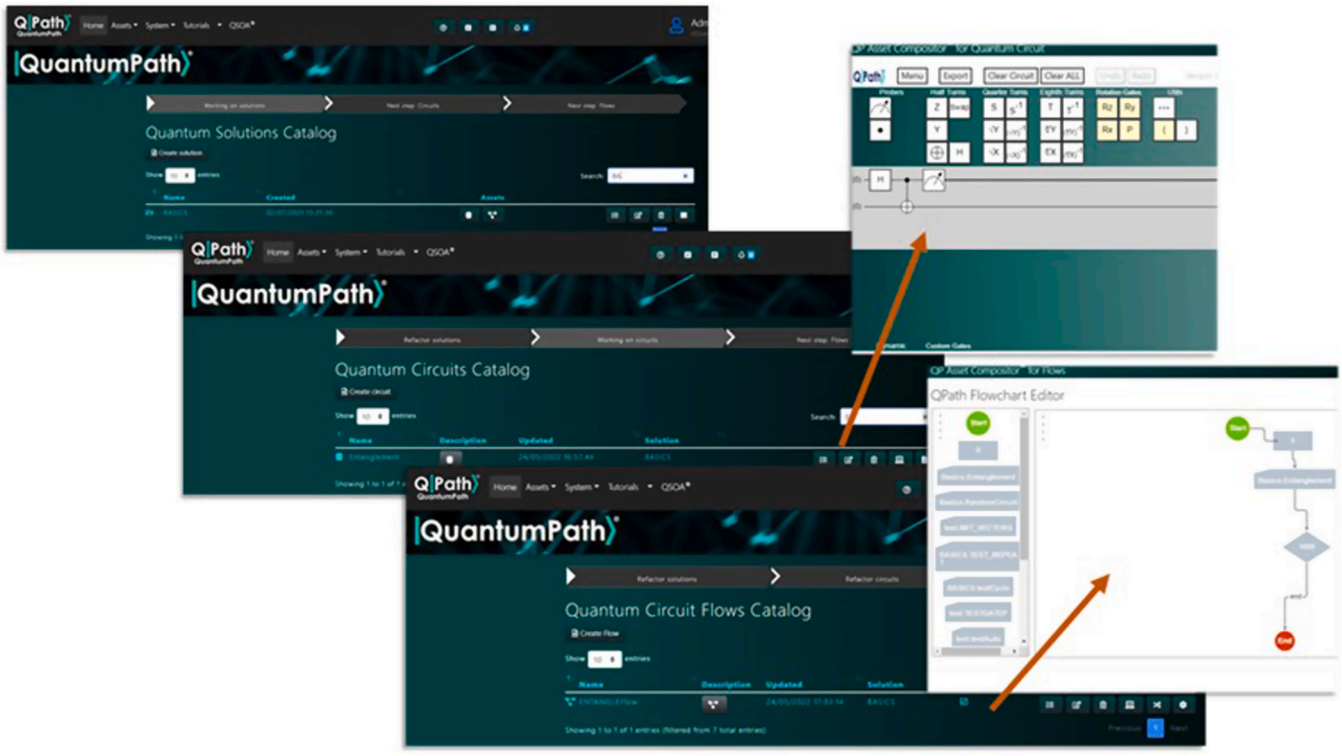


Fig. 14. QuantumPath® Entanglement Demo Application.

published "use cases" are summarized as follows:

Primitive	Purpose
GetQuantumSolutions	Allows the retrieval of the solution catalogue dictionary.
GetQuantumDevices	Allows the retrieval of the dictionary of the catalogue of quantum devices, linked to a solution (idSolution), required as a parameter.
GetQuantumFlows	Allows the retrieval of the dictionary of the catalogue of flows published in a solution (idSolution), required as a parameter.
RunQuantumApplication	Starts a labelled execution loop (aIterationName) of a certain flow (idFlow) linked to a solution (idSolution) on a quantum device supported by the solution (idVendor), required as parameters. Returns the execution token associated with the asynchronous job assumed by the QuantumPath® backend platform.
GetQuantumExecutionResponse	Allows the retrieval of the result of a specific execution cycle (executionToken) associated with the flow (idFlow) of a solution (idSolution), required as parameters. It will return a QER object with the information resulting from the execution.
GetQuantumExecutionHistoric	Using the filters, this method returns historical quantum execution information. If a filter is null, the system will return all of the possible results associated with this filter. That service makes possible the study of the quantum data and all of its metadata associated (including execution time)
GetQuantumExecutionHistoricResult	Return a specific record in time of a specific execution instance. This makes possible two relevant contexts in the democratizacion principle: reduce executions recovering previous executions as first advantage. Reover specific data when massive data are being analyzed, as second advantage.

Thanks to these primitives, classic clients can retrieve the elements associated with their user accounts and, based on the different dictionaries obtained, select the assets they want to execute and determine how. At any time from the QuantumPath® platform, these elements can be refactored and extended over time, with a very controlled impact on the integration layer.

The sequence diagram for a classic client to launch an experiment is thus shown in Fig. 13.

Of course, retrieving the dictionaries will allow the client to generate a "database" that can be reused in different use cases. In the qSOA® REST API, the response dictionaries are in the dictionary-type JSON format, which makes it extremely easy to transform the results into high-level objects with deserialization mechanisms specific to each programming language.

The same principle will be applied to the rest of the primitives. For the execution of a specific quantum use case, we will start from the context of having defined a “BASICS” solution in QuantumPath®, with a circuit that demonstrates the entanglement between qubits called “Entanglement”, and a flow that allows us to run it 1000 times to obtain a histogram of results (Fig. 14) and the solution linked to a set of selected quantum gate devices. In this case, we have created the quantum assets using the UI Visual “no-code” features of the product.

We will establish the context now. From QuantumPath® we obtain the identifiers of each asset to simplify the exercise, in such a manner that the identifiers[2] of each element are:

- “BASICS” solution, identifier “1″ -we can use the REST API GetQuantumSolutions to obtain the desired value.
- Flow “EntanglementFlow”, identifier “11″ -we can use the REST API GetQuantumFlows(idSolution) to obtain the desired value.
- Device. From the linked Quantum Devices to the Solution asset, we can choose the one we want to execute to our flow using

[2] The IDs are appropriate to the text. In the production environment they may differ. We can query these values using the visual environment or using the qSOA® appropriate methods.

Fig. 15. Sequence diagram for an Execute qSOA REST API action.

GetQuantumDevices(idSolution) REST API. In this case, we want to use the Qiskit local simulator of QuantumPath® backend, with code "1".

Example: We have also associated the devices associated to the solution of id "1", so that to recover the quantum devices, we can do the following:

HTTP property	Value
Method	GET
URL	https://qsoa.quantumpath.app:8443/api/connectionPoint/GetQuantumDevices/<<idSolution>> for example https://qsoa.quantumpath.app:8443/api/connectionPoint/GetQuantumDevices/1
Authorization	Bearer Token Authentication
Response	{ "1": "Q# Local Simulator Framework", "2": "QISKIT Local Simulator", "6": "IBM Cloud Quantum 32qbits simulator", "8": "QuTech Cloud 26qbit Single-Node Simulator", "10": "IBM Cloud Q_Santiago 5qbits computer", (*continued on next column*)

(*continued*)

"14": "AMAZON BRAKET 25qbits Local Simulator", "15": "AMAZON BRAKET State Vector simulator SV1", "26": "AMAZON BRAKET Rigetti Aspen 1 80qubits", "27": "AMAZON BRAKET IonQ 11qubits", "28": "AMAZON BRAKET Oxford Lucy 8qubits" }

Therefore, to execute the selected "quantum bussines use case", the sequence diagram could be the one shown in Fig. 15.

Hence, the qSOA API needs a few lines of code to execute a Quantum Use case. Regarding the implementation, of course, all telemetry and performance information will be available within the platform. All this information can be accessed both visually and programmatically (Fig. 16).

It is important to consider that some devices, depending on the complexity, waiting queues and other issues, can take anywhere from milliseconds to—in very specific cases—almost 1 hour. In the extreme case of having only one quantum computer of a certain model, it can even take days to find a slot for execution. It is for this reason that the asynchronous and decoupled model of qSOA® provides a sustainable execution framework. Since QuantumPath® stores the results in its backend, when new cases are executed, a classic system may involve recovering analysing the responses of previous executions.

Of course, the SDK of QuantumPath® or our own developed proxy classes, could have this asynchronous principle in mind to provide methods to run the quantum use cases in synchronous or asynchronous ways. Welcome to the Software engineer principles!

3.3. *Dynamic asset extension primitives*

Through this group of primitives, the qSOA® REST API allows the level to be further raised in terms of integration services with the classic layer, addressing problems related to the magnitude of the qubit resolution or the complexity of quantum circuits.

Additionally, it is important to note that QuantumPath® deals with two levels of intermediate language representation in its CORE. The following are two specifications that will be necessary to know for

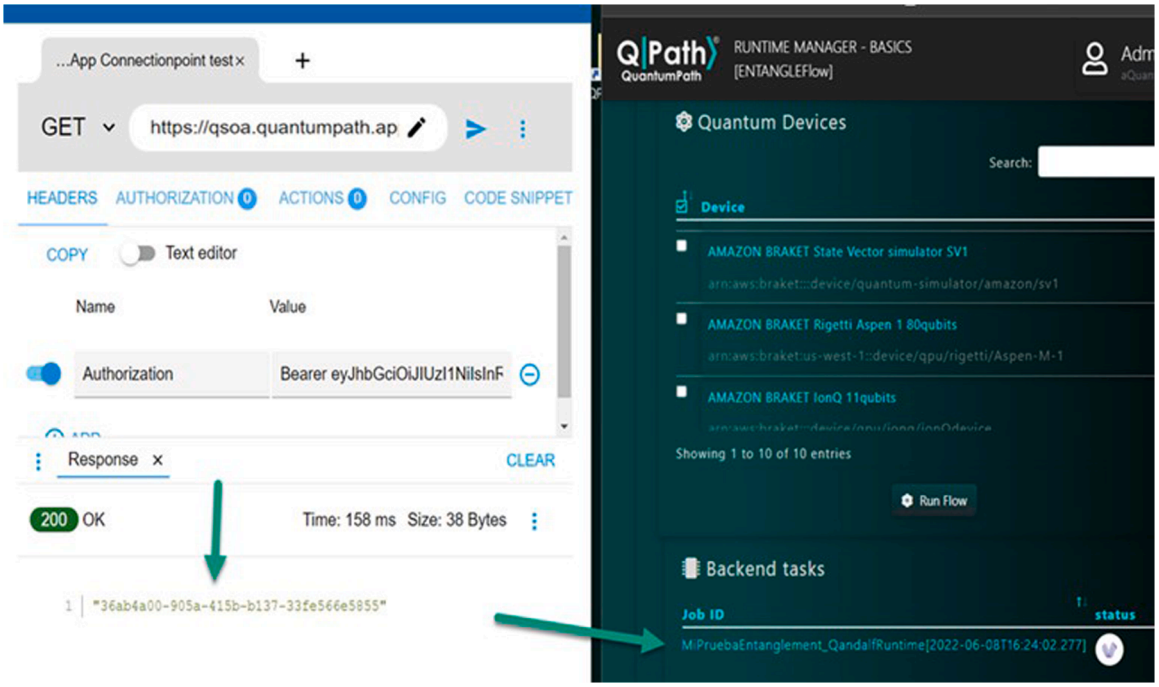


Fig. 16. The life cycle can be tracked and controlled at any time from QuantumPath®.

qSOA®:

- Visual Language (VL), which allows interaction with the representation for the visual tools of the system to be maintained.
- Intermediate Language (IL), which is the product of compiling the quantum assets. This product, the most important, is the one that the system will transpile according to the quantum devices linked to a solution.

This level of interaction with the QuantumPath® CORE facilitates the creation of new assets from the integration REST API, allowing the platform itself to internally manage the life cycle of the generated assets (their status, their compilation and its transpilation). In other words, qSOA® will automatically treat the internal processes of the *pipeline* associated with the quantum asset. For example, if a new quantum circuit is created with qSOA® in intermediate language, it will be automatically validated and automatically represented in the visual intermediate language.

All assets generated with qSOA® can thus be managed by the visual tools of QuantumPath® throughout their entire lifecycle. Of course, this new level of interaction means a higher level of complexity in terms of integration with a classical system, but still, it makes the process so natural that the complexity really lies in the dynamic use case itself.

The primitives available in the qSOA® REST API that allow automating the quantum life cycle pipeline are the following:

Primitive	Purpose
GetAssetCatalog	It allows retrieving the catalogue of quantum assets of a solution, based on its type of asset (Circuit or Flow) and its level of representation (VL or IL). It differs from **GetQuantumFlows** in that the former returns a dictionary of solution flows, while this primitive returns an object model with the key data of the asset: name, *namespace,* description, body, where the body, determined by the level of representation, can be visual language or intermediate language.
GetAsset	It allows retrieving a specific instance of a certain asset, determined by its ID based on its type (circuit or flow) and its representation level (VL or IL).
CreateAsset	It allows the creation of a new quantum asset created in a solution and initiates the asynchronous process of automatic lifecycle control.
UpdateAsset	It allows a certain quantum asset from a solution to be updated and starts the asynchronous process of automatic life cycle control.
GetAssetManagement results	When assets are created/updated, it is possible to know the status of the automatic life cycle process, processed in the *background,* carried out by the QuantumPath® CORE.
DeleteAsset	It allows a specific quantum asset from the solution to be eliminated based on its type (circuit or flow).

As an example, the HTTP call to create a quantum asset of type circuit could be:

Primitive	Purpose
HTTP POST	https://qSOA.quantumpath.app:8443/api/connectionPoint/CreateAsset?aSolutionID=idSolution

BODY	{
	"AssetID": −1,
	"AssetName": "EntanglementDynamicFromIL",
	"AssetNamespace": "Basics",
	"AssetDescription": "Entanglement demo qSOA v2",
	"AssetBody": "SCgwLCliKTtDTk9UKDAsMSwiIik7TSgwLCliKTs=",
	"AssetType": "GATES",
	"AssetLevel": "IL",
	"AssetLastUpdate": ""
	}

Where the *assetbody* is the Base64 representation of an IL circuit, for example:

H(0,"");CNOT(0,1,"");M(0,"");

3.4. Historic data query for analysis purposes

It is critical to handle the information generated after an execution. Study the data results and its telemetry is mandatory to manage the "quantum knowledge" appropriately. Even more, we are not talking about the information just calculated after the execution, we are talking about the information over time. So, we can manage intelligence over raw data. This capability is exposed as a service in qSOA®.

There exist two API functions to query the historical execution data:

Primitive	Purpose
GetQuantumExecutionHistoric	It allows the query of multiple datasets of quantum executions and their results, based on the filter selectors. The possible filters can be ids of assets (solution or flow or device), timestamps, type of result as well as kind of target. If the filter is not specified or null, it will act as a wildcard.
GetQuantumExecutionHistoricResult	It allows to recover a specific record of quantum execution.

As an example of a query:

Primitive	Purpose
HTTP POST	https://qSOA.quantumpath.app:8443/api/connectionPoint/GetQuantumExecutionHistoric/10,111/null/null/638,022,338,700,000,000/null/null/true
BODY	[{ "IdResult": 20,756, "IdSolution": 10,111, "SolutionName": "aQAOA_TEST", "IdFlow": 112, "FlowName": "MOCHILAFlow", "IdDevice": 20, "DeviceName": "qasm_simulator", "DeviceShortName": "IBM QAOASim*", "DeviceVendor": "IBM", "IsLocalSimulator": true, "DeviceTypeName": "QUANTUM ANNEALING", "ResultHistogram": "{\"aQAOA_TEST_10,111_aQAOA_MOCHILA_1_0\": {\"number_of_samples\": \"29\", \"shots\": \"1000\", \"number_of_variables\": \"5\", \"sample_energy\": \"−1.3999999999999986\", \"sample_occurence\": \"1\", \"fullsample\": {\"Boxes[1]\": \"1.0\", \"Boxes[2]\": \"1.0\", \"Boxes[3]\": \"1.0\", \"Boxes[4]\": \"0.0\", \"Boxes[5]\": \"0.0\"}}}", "ExecutionDate": "2023–03–13T13:52:17.06″, "DurationMinutes": 0.39662034, "ResultType": "OK", "ResultDescription": "" }, { "IdResult": 20,755, "IdSolution": 10,111, "SolutionName": "aQAOA_TEST", "IdFlow": 112, "FlowName": "MOCHILAFlow", "IdDevice": 35, "DeviceName": "AWS_LocalSimulator_Gates aQAOA", "DeviceShortName": "AWS Gates Sim* QAOA", "DeviceVendor": "AWS", "IsLocalSimulator": true, "DeviceTypeName": "QUANTUM ANNEALING", "ResultHistogram": "{\"aQAOA_TEST_10,111_aQAOA_MOCHILA_1_0\": {\"number_of_samples\": \"50\", \"shots\": \"1000\", \"number_of_variables\": \"5\", \"sample_energy\": \"−1.3999999999999986\", \"sample_occurence\": \"131\", \"fullsample\": {\"Boxes[1]\": \"1\", \"Boxes[2]\": \"1\", \"Boxes[3]\": \"1\", \"Boxes[4]\": \"0\", \"Boxes[5]\": \"0\"}}}", "ExecutionDate": "2023–03–13T13:52:05.947″, "DurationMinutes": 0.21122784166666667, "ResultType": "OK",

(*continued on next page*)

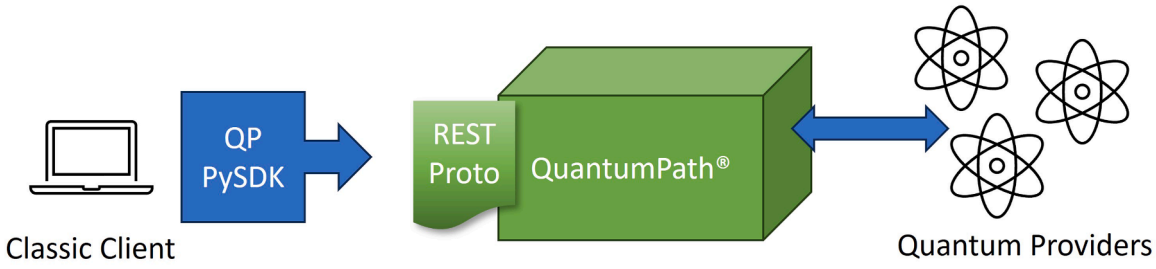


Fig. 17. Deployment model of QuantumPath® qSOA.

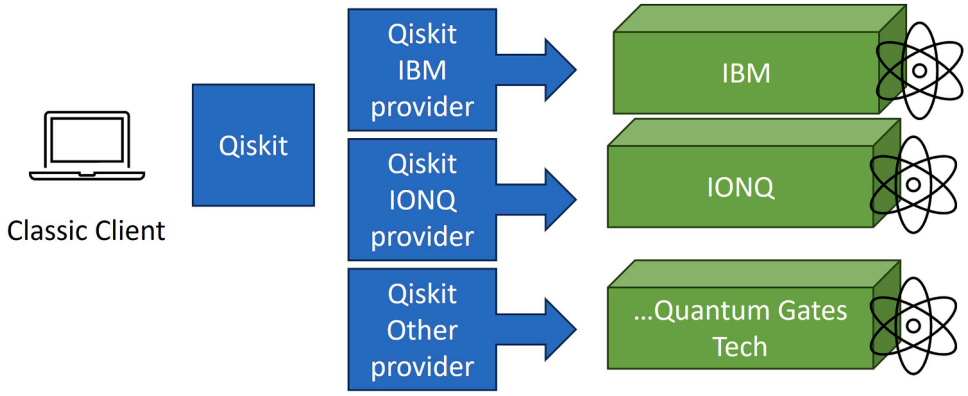


Fig. 18. Qiskit client model provider.

(*continued*)

Primitive	Purpose
	"ResultDescription": "" }, …]

In this example, we have used the filters to answer a query like: "give me all the executions for the solution *aSolutionID*, from date *aTimeStamp* that have a result of OK". Taken all the metadata it will be really easy compose an analysis datamart to study time evolution, trends, and predictions.

4. The quantumpath qSOA® sdk for python (PySDK)

The API REST provided by qSOA® services, are technology agnostic. It's an open text-based protocol programming language and/or development platform independent. You can use any language with communication capabilities to make the calls. But, of course, this reduce the productivity when you start a project because you need to implement some kind of library to encapsulate the REST API and provide a high-level interface to be used by the team in it's favourite language. At this point, certain commodities could be implemented too to make more feasible the library: classes for modelling the circuits without potential syntax errors, classes to encapsulate the results… etc.

For that reason, QuantumPath® designed as a business-oriented product, provide up to two prebuilt specific languages SDK implementations (and growing) that contains all the API REST services implemented and a complete set of value-added capabilities -that acts as those commodities previously mentioned- added. The language selection responds to criteria such as the massive implantation in IT ecosystems. The first development languages selected has been Python and .NET. Languages such Java and Javascript are being developed.

The PySDK of QuantumPath® is an example of a set of classes that act as proxy components provided to explore and execute the qSOA REST API from a high-level language. The PySDK is a Python 3.xx package, which can be obtained from the main PyPI repository.[3] It provides all the necessary abstractions, so it is not necessary to know the low-level REST API. With Python objects, it will be possible to establish a working context with the platform's services and have access to practically any use case needed:

- Create complete classic applications that require the launch of quantum use cases as part of their logic.
- Create classic applications that dynamically generate the quantum resources they need to run more complex and adaptive algorithms.
- Add to the created classic logic analytical capabilities to study the quantum execution historical data and its telemetry.

As QuantumPath® is agnostic of the quantum technology and provider, we only need to deploy in the work machine one SDK: PySDK. It's not necessary to deploy as many SDKs as there are providers we want to use (Fig. 17). This is a great differentiation comparing to other products. For example, if you want to use Qiskit to work with different quantum providers you will need to deploy as many extensions as providers you want to use (Fig. 18). At the same time, if you want to create an annealing problem, you will need to design the problem totally from scratch assuming all the elements, while qSOA® only requires the use case name or the formal IL definition. And, even, not worrying about the destination quantum computer. It is also different of technologies like AWS Bracket, where there exists a reduction in the way you use different providers in terms of SDK deployments. In Braket you must be aware about the right set of gates to be used when you select one quantum hardware.

In the Python SDK quantum devices and flows are three main elements when creating use cases on the QuantumPath® platform. Using QuantumPath® PySDK, three methods are provided in each of the cases

[3] https://pypi.org/project/QuantumPathQSOAPySDK/. More info can be obtained through the https://qsoa.quantumpath.app:8443 help and tutorials.

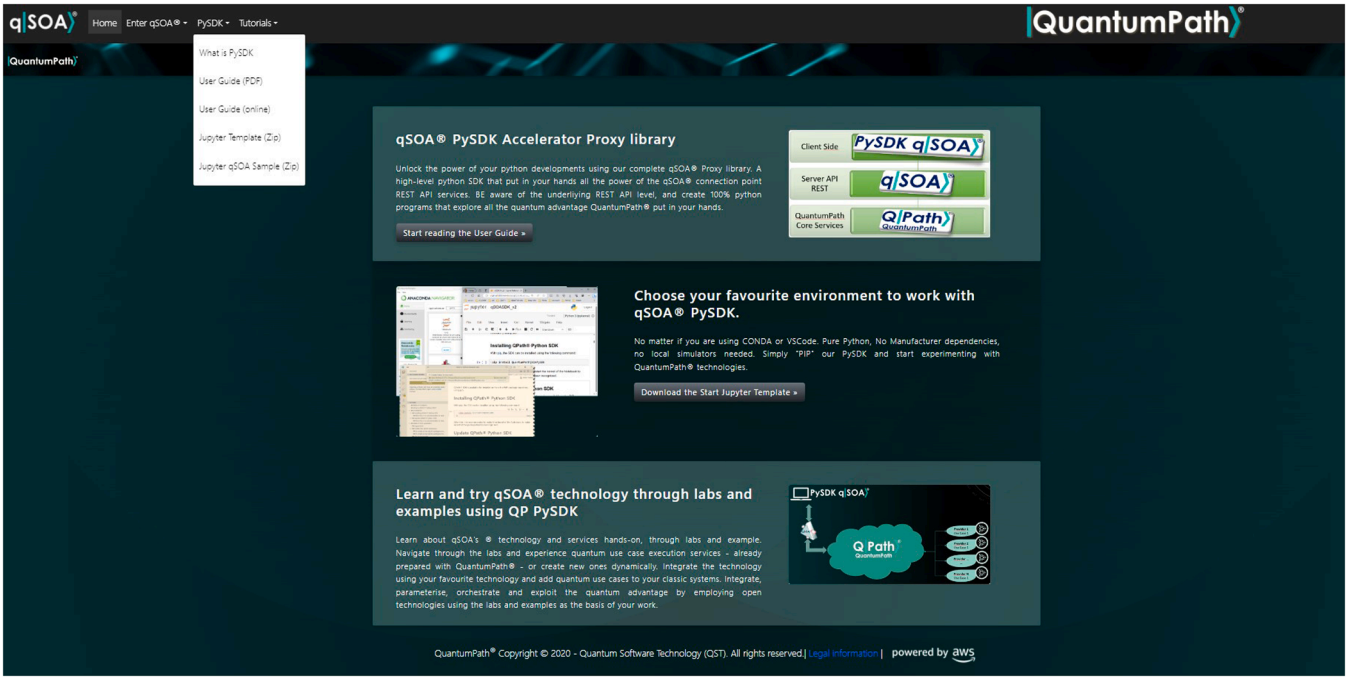


Fig. 19. qSOA Web portal.

to obtain information from these. Returning the raw information as a dictionary for easy reading and processed if we want to have the information in that state; together with two other options that act as "commodities", to obtain an own object that contains the data, or just the name of the element searched for in the form of a text string.

The qSOA web portal (Fig. 19), provides a lot of information and resources to start from scratch. Online SDK description, PDF user manual and a complete set of tutorials provide the necessary for learn fast the technology and start to build complete solutions using the SDK. Step by step or go to specific knowledge.

5. Examples

In this section we show different examples of qSOA® use

5.1. Example 1: API template, launch quantum use cases in hybrid architectures

In this example, we show a templated Python program (created from one of the templates offered in the qSOA® resources) to execute a user-selected quantum business case by querying the system and the standard input console. This example demonstrates the API LEVEL 1, execution of Quantum use cases developed with QuantumPath® engineering tools and stored as an asset:

```
# IMPORT SDK
from QuantumPathQSOAPySDK import QSOAPlatform
# Connect explicitly
username = input('Username: ')
password = input('Password encrypted in SHA-256: ')
qsoa = QSOAPlatform(username, password)
# QUERY AND SELECT A SOLUTION DICTIONARY WITH ID
solutionList = qsoa.getQuantumSolutionList()
print('Solution list:', solutionList)
idSolution = int(input('Choose Solution ID: '))
# QUERY AND SELECT SOLUTIONs FLOW DICTIONARY
flowList = qsoa.getQuantumFlowList(idSolution)
print(f'Solution {idSolution} flows:', flowList)
idFlow = int(input('Choose Flow ID: '))
# QUERY AND SOLUTIONs Linked DEVICES
deviceList = qsoa.getQuantumDeviceList(idSolution)
print(f'Solution {idSolution} devices:', deviceList)
idDevice = int(input('Choose Device ID: '))
# RUN FLOW –USE CASE–, SYNC WAY
applicationName = input('Application name: ')
application = qsoa.runQuantumApplicationSync
(applicationName, idSolution, idFlow, idDevice)
# GET RESULTS WITH APPLICATION OBJECT
execution = qsoa.getQuantumExecutionResponse
(application)
# EXECUTION OBJECT INFORMATION
print('Exit code:', execution.getExitCode())
print('Exit message:', execution.getExitMessage())
print('Solution name:', execution.getSolutionName())
print('Flow name:', execution.getFlowName())
print('Device name:', execution.getDeviceName())
print('Histogram:', execution.getHistogram())
print('Duration:', execution.getDuration())
```

When running a quantum gates flow or a quantum annealing flow, the results can be printed in two possible ways:

```
# REPRESENT GATES RESULTS
qsoa.representResults(execution)
```

or

```
# REPRESENT ANNEALING RESULTS
annealing_representation = qsoa.representResults
(execution_annealing)
print(annealing_representation)
```

In this example, it can be seen the reduce number of lines needed to execute and get results of a quantum use case, and the sections in which decisions can be taken and code be written to select the parameters needed to run the program. This code could be refactored in other frameworks like PHP, console or Jupyter notebooks easily.

The results format depends on the quantum technology selected, but at the end are JSON structs with unified format that can be easily extracted and post-processed in the classic client to transform into user information. But again, let us put the note: this result is binary independent of the quantum provider used. Again, here we have an advantage through the rest of competitors.

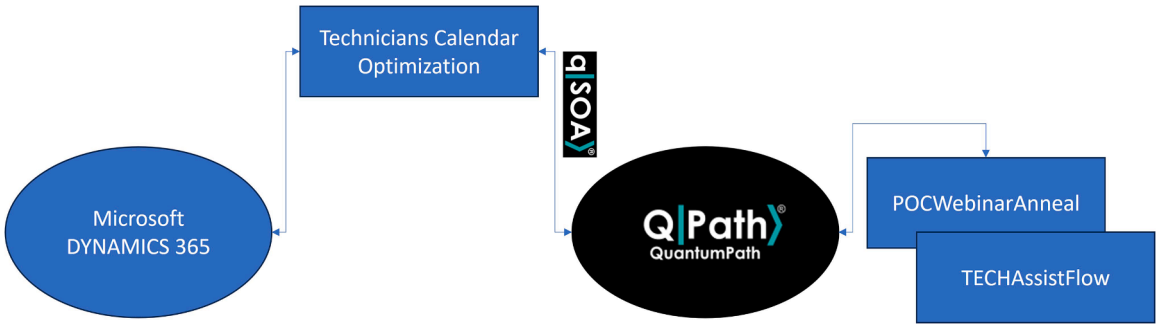


Fig. 20. Example of qSOA® integration with a classical CRM.

5.2. Example 2: API template, dynamic quantum use cases in hybrid architectures

In this example, we need to create some kind of circuit from dynamic information generated in the classic side or modify an existent circuit that must be refactored to adapt to a new situation.

In this case, the base template of the quantum program that creates a circuit could be like the example of the section 4.1 but creating before run the use case:

```
# IMPORT SDK
from QuantumPathQSOAPySDK import QSOAPlatform
# Connect explicitly
username = input('Username: ')
password = input('Password encrypted in SHA-256: ')
qsoa = QSOAPlatform(username, password)
# QUERY AND SELECT A SOLUTION DICTIONARY WITH ID
solutionList = qsoa.getQuantumSolutionList()
print('Solution list:', solutionList)
idSolution = int(input('Choose Solution ID: '))
# CIRCUIT BODY (WITH METHODS)
circuitBody = qsoa.CircuitGates()
circuitBody.h(0)
circuitBody.cx(0, 1)
circuitBody.measure()
#OTHER WAY TO CREATE THE CIRCUIT, Using IL string
#circuitBody = '''INIT(ZERO,ZERO);H(0,"");CNOT(0,1,"");M(0,"")|M(1,"");'''
# CREATE CIRCUIT ASSET
circuitName = 'BellPair_Dynamic'
circuitNamespace = 'Samples.Gates'
circuitDescription = 'Dynamic BellPair quantum gate circuit'
circuitType = 'GATES'
circuitLevel = 'IL'
assetManagementData = qsoa.createAssetSync(idSolution, circuitName, circuitNamespace, circuitDescription, circuitBody, circuitType, circuitLevel)
# FLOW BODY (WITH METHODS)
flowBody = qsoa.CircuitFlow()
startNode = flowBody.startNode()
initNode = flowBody.initNode(0)
circuitNode = flowBody.circuitNode(f'{circuitNamespace}.{circuitName}')
repeatNode = flowBody.repeatNode(100)
endNode = flowBody.endNode()
flowBody.linkNodes(startNode, initNode)
flowBody.linkNodes(initNode, circuitNode)
flowBody.linkNodes(circuitNode, repeatNode)
flowBody.linkNodes(repeatNode, endNode)
#OTHER WAY TO CREATE THE FLOW, Using IL string
#flowBody= f'''ABSTRACT(START,);REPEAT(0|100,CIRCUIT({circuitNamespace}.{circuitName}));ABSTRACT(END,);'''
# CREATE FLOW ASSET, By Default Published to be used by qSOA Clients
flowName = 'BellPair_Dynamic_Flow'
flowNamespace = 'Gates'
flowDescription = 'Dynamic BellPair Flow'
flowType = 'FLOW'
flowLevel = 'IL'
assetManagementData = qsoa.createAssetSync(idSolution, flowName, flowNamespace, flowDescription, flowBody, flowType, flowLevel, True)
idFlow = assetManagementData.getIdAsset()
#
# At this point, repeat the code of example 1. Execute use case.
# but using the Id of the products created
#
# QUERY AND SOLUTIONs Linked DEVICES
deviceList = qsoa.getQuantumDeviceList(idSolution)
print(f'Solution {idSolution} devices:', deviceList)
idDevice = int(input('Choose Device ID: '))
# RUN FLOW –USE CASE–, SYNC WAY
applicationName = input('Application name: ')
application = qsoa.runQuantumApplicationSync(applicationName, idSolution, idFlow, idDevice)
# GET RESULTS WITH APPLICATION OBJECT
execution = qsoa.getQuantumExecutionResponse(application)
# REPRESENT GATES RESULTS
qsoa.representResults(execution)
```

Observe the changes of the main program and you could see the coherent way to extend a program with more features provided by the SDK. Of course, to change the type of quantum program will be as easy of select a solution that supports the annealing technology, change the type of the Circuit class and set a circuit body following the IL Rules of an annealing problem and run the experiment in the quantum providers associated with the solution (that could be both native annealing devices and QAOA quantum gates providers supported).

5.3. Example 3: integrating well-known products with quantum technologies

Let us imagine a calendar optimization algorithm to organize technicians in a company, so as to distribute them into "bubbles" to avoid virus propagation. Let us suppose there are different technicians: Human Resources, Sales, and IT people, with the following rules:

- Sales and IT people can never stay in the office on the same day
- Human Resources people cannot stay in the office on consecutive days

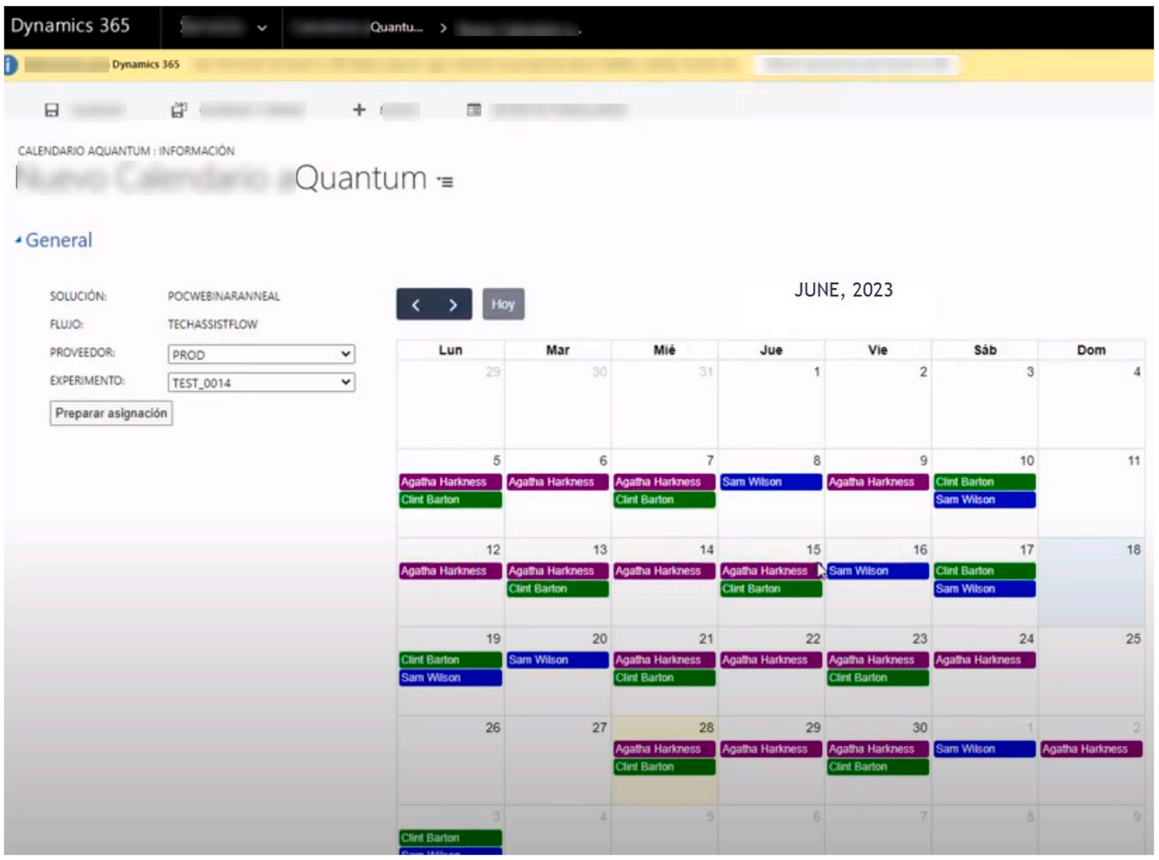


Fig. 21. The client side.

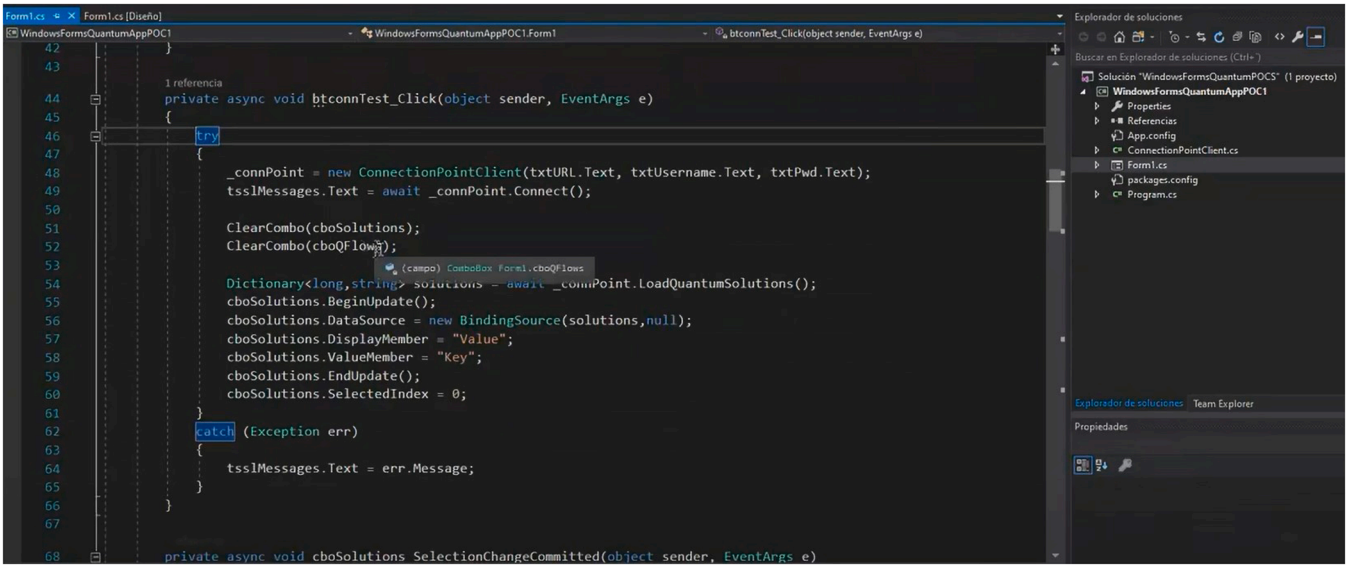


Fig. 22. The qSOA .NET SDK client layer.

And the goal is to maximize IT assistance. In this case, we will use (Fig. 20) Microsoft Dynamics 365 on-premises, a quantum solution called "POCWEBINARAnneal", which codifies the quantum optimization algorithm, and a flow ("TECHASSISTFlow) that executes the annealing solution.

In Dynamics 365 we can have as an extension the optimisation of calendar organisation acting as a client. QPath attends the connection points activations, prepares the job, and executes it with the transpilation of the assets and calls the hardware provider (in this case, for example, d-Wave). The hardware provider does the job, gets the results, and then, results are consolidated into the platform, and returned to the client. The CRM client recovers the data and populates the calendar.

In the demonstration, you could see the best practices of a complete hybrid classical-quantum solution isolated in clear layers such:

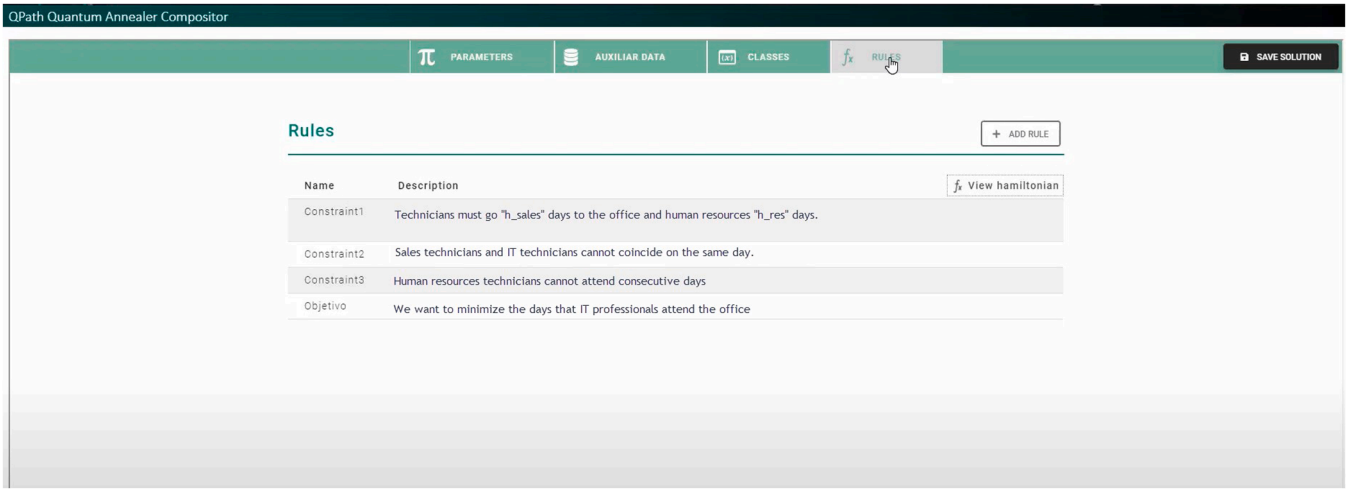


Fig. 23. The QuantumPath® service layer.

- The client layer, in these case, Microsoft CRM 365 (Fig. 21)
- The client code, using qSOA .NET SDK (Fig. 22)
- The quantum use case designed and stored under the QuantumPath® engineer life cycle tools (Fig. 23).

The complete demonstration is explained in a YouTube webinar,[4] where another example of qSOA® use is also explained using the case of a quantum number generator integrated with Microsoft Navision.

6. Caracterization of qSOA®

6.1. Comparison with alternative technologies

As we mentioned earlier, qSOA® technology offer some advantages of being practical at an industrial level since it provides:

- Seamless integration of quantum services with classical systems, making it suitable for commercial adoption. We clearly define the "quantum" use case that will be exploited from classic systems and not as a general purpose one, that is more focused on the proof of concept or monolithic modules. So, we separate in a clear way what is the client responsibility and what is quantum layer responsibility. Thanks to this separation, the quantum specialist knows where to put the focus in each layer. In other technologies like Qiskit, Bracket, d-Wave leap… they mix in the same place every element of the solution. By default, they are mixing the classical part with the quantum part, making more difficult the integration decisions in industrial designs.
- Dynamic asset extension, allowing real-time creation and management of quantum assets, making it adaptive to changing problems. Because we separate as clear abstractions the classical side and the quantum side, we provide clear quantum abstractions that can be easily integrated in classical sides. And each of these abstractions has its own life cycle and responsibilities in terms of computer resources and architectural designs.
- A high-level API REST open and well documented protocol interface. This makes it possible the language agnostic advantage of the technology. Everyone can create its own API REST implementations using its own favourite language. We provide self-created SDK libraries implemented in the more common programming languages in professional environments like Python, .NET, Java and JavaScript. For example, Qiskit today only provides a Python oriented implementation of its services over an open web protocol that is not their priority to be documented and forcing the providers that wants to extends their SDK to use just this unique language. In our side, we provide multiple programming languages to create the client layer and the common language used to represent the quantum assets is a metalanguage -not functional by itself- 100 % agnostic of the platform.
- An API REST without any dependencies with third party quantum components. The client layer only needs to implement a connection to the REST API to access the agnostic technology of QuantumPath®. Consequently, the SDK created on the top of this REST API does not have any dependencies with third-party SDK or components. If you want to use the QP Python SDK, you only need to have the proper Python environment ready and deploy the QuantumPath® PySDK. Qiskit, to be used with IBM Quantum or other supported partners, need the download of multiple packages as dependencies.
- A set of gates and rules composition that can be used in every provider that supports the quantum technology to be applied. This applies to the results format as well. Qiskit and its multiple provider support, obliges the user to keep in mind the limitations of every "supported" platform selecting the gates set or annealing elements as well as the binary format representation returned.

6.2. Practical implementation challenges

In the implementation of projects using qSOA® we have found the following challenges:

- Parallelize the development tasks between quantum algorithms team needs and classical IT engineers team needs. The challenge was to define what is the convergence point. In this context, qSOA® makes really easy to make decisions about how to separate the roles and the tasks. While the IT engineers creates the solution with the use cases defined, the quantum team create, test, and refactor the solution in isolation. The IT engineers, experts in .NET framework, can use QP . NET SDK for qSOA® to access to the quantum uses cases.
- In the example of Section 5.3, we face a solution to integrate an optimization algorithm for a resources calendar over Microsoft CRM 365 that need to access to a quantum developed use case through the microservices Javascript mandatory language. qSOA® API REST make possible the integration between these two products. In a refactorization of this case, we added several dynamic updates to the quantum use case to modify the Hamiltonian definition under certain business rules.

[4] https://www.youtube.com/watch?v=Vo7atNN0AAg

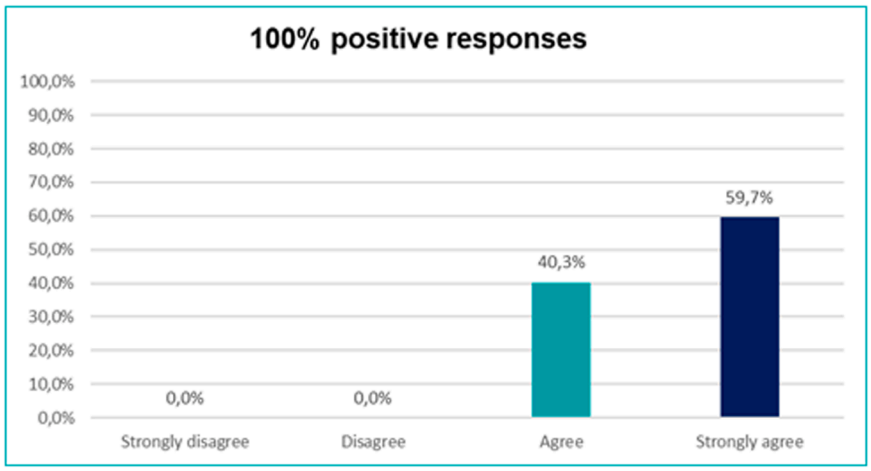


Fig. 24. Results to the question: "qSOA® simplifies the integration of quantum software and classical IT?".

- When training researchers and students, qSOA® makes it easier to explain the concepts and relevance of the quantum advantage through Jupiter Python notebooks. PySDK simplifies at the top the deployment of the working environment and clearly isolates the concepts, so the training session maximize its productivity, putting the focus on the specific matter isolating it from platform details, operating system requirements, quantum provider specific details as examples of elements that affect the training process.

6.3. *Survey about the usefulness of qSOA®*

On the other hand, we held a workshop "Introduction to Amazon Braket and QuantumPath®: Creating Quantum Solutions" carried out April 27th-28th 2022, with the collaboration of AWS Braket, with 200 attendees who were able to test both platforms for 72 h. A survey (QuantumPath, 2022) was elaborated to collect the users' opinion about the actual functioning and features of QuantumPath®. Responses were obtained from the attendees from Italy, Portugal, Brazil, Mexico, Chile, Uruguay, USA, France, and Spain. One of the questions dealt specifically with the use of qSOA® whose result is shown in Fig. 24.

7. Conclusions and future work

Quantum computers are proving their "advantage", or at least their "quantum utility" in the sense of efficiency and practicality over classical machines of similar size, weight and cost (Herrmann et al., 2023). However, the good principles of computing do not change we need access via a communications network, an access protocol to the services of the new system, and we need to apply best practices at all levels of access (security, traceability, governance…) that have already been in place and well-tested for a long time.

In this article we describe qSOA®, a technology that sits at the service layer of the computer system and allows the quantum services platform to be exploited by means of an API. This API provides a set of well-defined functions with which a classical system can access quantum products as if they were just another piece of classical software, accessing the use case and not its complexities. From qSOA® it is also possible to interact with the quantum platform to define new quantum products in a totally dynamic way and adapted in real time to existing problems.

We are convinced that the practical adoption of quantum computing in the current technological context is a hybrid one, so we propose qSOA® as a clear path to address the incorporation of quantum technology to classical computing in a reliable, fast way, based on the best principles of computer engineering.

We will continue to evolve qSOA® as the QuantumPath® platform will evolve. We will provide better and new services, and new languages SDK. Of course, one of the biggest challenges that must be covered will be the treatment of massive data to be processed by quantum algorithms in the moment that the qubits scale made it possible to make this step feasible for the enterprise context.

CRediT authorship contribution statement

José Luis Hevia: Conceptualization, Methodology, Software, Writing – review & editing. **Guido Peterssen:** Conceptualization, Methodology, Software, Writing – review & editing. **Mario Piattini:** Conceptualization, Methodology, Software, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Ali, S., Yue, T., Abreu, R., 2022. When software engineering meets quantum computing. Commun ACM 65 (4), 84–88.
Akbar, M.A., Khan, A.A., Rafi, S., 2023. A systematic decision-making framework for tackling quantum software engineering challenges. Autom. Softw. Eng. 30, 22.
Barbosa, L.S., 2020. Software engineering for 'quantum advantage. In: In 2020 IEEE/ACM First International Workshop on Quantum Software Engineering (Q-SE). ICSEW'20, May 23–29, 2020. Seoul, Republic of Korea, pp. 427–429.
IBV, Institute for Business Value, 2021. The quantum decade. A playbook for achieving awareness, readiness, and advantage. IBM Institute for Business Value. https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/quantum-decade.
GAO, 2021. Technology assessment. quantum computing and communications. status and prospects GAO-22-104422, United States Government accountability office. Report to congressional addressees, October 2021. https://www.gao.gov/products/gao-22-104422.
Herrmann, N., Arya, D., Preis, F., Prestel, S. Doherty, M.W., Mingare, A., and Plillay, J. (2023). arXiv:2303.02138v1 [quant-ph] 3 Mar 2023.
y Hevia, J.L., Peterssen, G., Ebert, C., Piattini, M., 2021a. Quantum Computing. IEEe Softw. 38 (5), 7–15.
y Hevia, J.L., Peterssen, G., Piattini, M., 2021b. QuantumPath: A quantum software development platform. Software Practice Experience. https://doi.org/10.1002/spe.3064.
Moguel, E., Rojo, J., Valencia, D., et al., 2022. Quantum service-oriented computing: current landscape and challenges. Software Qual J 30, 983–1002.

y Pérez-Castillo, R., Serrano, M.A., Piattini, M., 2021. Software modernization to embrace quantum technology. Adv Eng Software 151, 102933. https://www.sciencedirect.com/science/article/abs/pii/S0965997820309790.

y Piattini, M., Peterssen, G., Pérez-Castillo, R., 2020a. Quantum computing: a new software engineering golden age. ACM SIGSOFT Software Eng. Notes 45 (3), 12–14. https://dl.acm.org/doi/10.1145/3402127.3402131.

Piattini, M., Peterssen, G., Pérez-Castillo, R., Hevia, J.L., et al., 2020b. The talavera manifesto for quantum software engineering and programming. In: QANSWER 2020 QuANtum SoftWare Engineering & Programming. Proceedings of the 1st International Workshop on the QuANtum SoftWare Engineering & Programming. Talavera de la Reina, Spain, pp. 1–5, 11-12 February 2020. http://ceur-ws.org/Vol-2561/paper0.pdf.

Piattini, M., Serrano, M., Pérez-Castillo, R., Peterssen, G., Hevia, J.L., 2021. Towards a quantum software engineering. IT. Prof. 23 (1), 62–66. https://doi.org/10.1109/MITP.2020.3019522. Jan.-Feb. 2021. https://ieeexplore.ieee.org/document/9340056.

QuantumPath, 2022. QuantumPath® evaluation in a real situation. The QPath Blog. https://www.quantumpath.es/2022/05/07/quantumpath-evaluation-in-a-real-situation/.

SEI, 2021. Architecting the future of software engineering. A National Agenda for Software Engineering Research & Development. Carnegie Mellon University, Software Engineering Institute. November 2021. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=741193.

y. In: Serrano, M., Pérez, R., Piattini, M. (Eds.), 2022. Quantum software engineering. Springer, Berlin, 2022.

Serrano, M., Cruz-Lemus, J.A., Pérez-Castillo, R., Piattini, M., 2023. Quantum software components and platforms: overview and quality assessment. ACM. Comput. Surv. 55 (8), 164:1-164:31.

WEF, 2022. State of quantum computing: building a quantum economy. insight report, September 2022, World economic forum.

Zhao, J., 2020. Quantum software engineering: landscapes and horizons. arXiv preprint arXiv:2007.07047.