Diseño de Servicios Cuánticos a través de la Especificación AsyncAPI

Jorge Casco Seco[1 [0009−0003−9166−6827]], Jaime Alvarado-Valiente[1 [0000−0003−0140−7788]], Javier Romero-Álvarez[1 [0000−0002−3162−1446]], Enrique Moguel[2 [0000−0002−4096−1282]], José García-Alonso[1 [0000−0002−6819−0299]], Carlos Canal[3 [0000−0002−8002−0372]], and Juan M. Murillo[2 [0000−0003−4961−4030]]

[1] Universidad de Extremadura, Escuela Politécnica, Quercus Software Engineering Group, Cáceres, España
jcascose@alumnos.unex.es, {jaimeav, jromero, jgaralo}@unex.es
[2] CénitS–COMPUTAEX, Cáceres, España
enrique@unex.es, director@cenits.es
[3] Universidad de Málaga, ITIS Software, Málaga, España
carloscanal@uma.es

Abstract. La computación cuántica ha evolucionado de ser una idea teórica a convertirse en una realidad tangible. Aunque no es posible acceder directamente a un ordenador cuántico de la misma manera que a los ordenadores convencionales, existen proveedores de servicios que ofrecen la posibilidad de utilizar esta tecnología. Sin embargo, estos servicios presentan ciertas limitaciones, especialmente la falta de herramientas que simplifiquen su uso y manejo. Este trabajo se enfoca en Amazon Braket, un servicio de computación cuántica ofrecido por Amazon, el cual procesa las solicitudes en colas de espera, lo que puede resultar en tiempos prolongados para obtener los resultados. Además, los resultados se almacenan en un servicio de almacenamiento proporcionado por el proveedor, lo que permite a los desarrolladores verificar el estado de las tareas cuánticas y recuperar los resultados obtenidos. No obstante, dado que el tiempo de finalización de la operación es incierto, este enfoque puede resultar ineficiente. Por lo tanto, se propone una solución alternativa que utiliza una arquitectura asíncrona e se integra con otro servicio, mediante una modificación de la especificación AsyncAPI. El objetivo es generar sistemas basados en eventos para integrar de manera programática los servicios cuánticos como parte de otros sistemas de software. Esto implica enviar el código cuántico a ejecutar al proveedor y obtener los datos resultantes de manera asíncrona.

Keywords: Computación Cuántica · Ingeniería de Software Cuántico · Servicios Cuánticos · AsyncAPI.

1 Introducción

La computación cuántica es un nuevo paradigma de computación que se basa en el uso de la mecánica cuántica. Al contrario que los ordenadores tradicionales,

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

estos computadores trabajan a escalas subatómicas. El principal beneficio que se obtiene con su uso es una mayor eficiencia a la hora de ejecutar algoritmos. Aunque los computadores clásicos puedan resolver estos problemas, no son capaces de obtener la mejor solución en un tiempo razonable, cosa que los computadores cuánticos sí pueden [1]. Esto es debido a la alta capacidad de cómputo que prometen y que puede ser utilizada de forma revolucionaria en diferentes ámbitos como la criptografía, la economía o la salud [2].

Además, los algoritmos que se ejecutan en un computador cuántico pueden beneficiarse de las propiedades específicas de este paradigma, como puede ser la superposición, la cual permite al *Qubit* estar en los estados 0 y 1 al mismo tiempo, obteniendo más combinaciones que con un bit clásico, o el entrelazamiento, que se da entre dos *Qubits* de forma que al modificar el estado de uno de ellos se modifica el otro de forma instantánea, facilitando la ejecución de operaciones de forma paralela. Los algoritmos cuánticos se representan como un circuito, que no es más que una sucesión de puertas lógicas cuánticas. Los circuitos son actualmente la forma en la que se programa los computadores cuánticos [1], aunque se está comenzando a trabajar en nuevas representaciones para agilizar el desarrollo y aumentar la abstracción [3], además de definir procesos de desarrollo, lo que constituye el embrión de una Ingeniería de Software Cuántico [4].

Independientemente de su potencia computacional, los sistemas cuánticos no reemplazarán completamente a los clásicos. Los costes de crear estas arquitecturas, unido a que existen muchos problemas que a día de hoy se pueden resolver de forma sencilla y eficiente utilizando computadores tradicionales, implica que los computadores cuánticos convivirán con los clásicos [5], mediante su integración en sistemas híbridos clásico-cuánticos. Pero no son todo ventajas, las herramientas que actualmente se pueden utilizar para lanzar estos servicios cuánticos funcionan a bajo nivel [6], por lo que los desarrolladores necesitan conocer cómo acceder y cómo programar cada computador cuántico.

La integración de servicios cuánticos y clásicos es la mejor forma de abordar estos problemas, permitiendo que los servicios clásicos invoquen a servicios cuánticos alojados en la nube [7]. Esta solución por ahora parece que se extenderá a largo plazo, ya que los proveedores de servicios cuánticos no muestran intención de modificar la forma de ofrecer dichos servicios.

Por otro lado, Amazon Braket, uno de los preveedores de servicios en el ámbito de la computación cuántica más conocido, indica en su documentación oficial que la manera de recuperar la información servida es utilizando espera activa hasta que se haya completado la tarea. Esto se aleja de los principios de la Ingeniería de Software. Por ello, este trabajo pretende resolver un problema intrínseco de plataformas de Computación Cuántica como Servicio, que es la recuperación de los resultados de forma eficiente.

Para darle solución a este problema, se propone que en lugar de enviar peticiones para recuperar el resultado, consumiendo tanto tiempo de CPU como de red sin hacer nada, se recibirá el resultado del computador automáticamente una vez completado. Para ello, se utilizará una arquitectura asíncrona. Aprovechando las funciones Lambda de AWS, se recuperará el resultado que Amazon Braket

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

haya almacenado en un *bucket* S3 haciendo uso de un único mensaje, sin saturar la red y sin necesidad de esperar a que se complete la tarea en Amazon Braket.

Para presentar esta propuesta, en las siguientes secciones del resto del artículo se indica la motivación para la creación de este proyecto, la arquitectura generada, el proceso de generación de servicios, un ejemplo de uso, trabajos relacionados y las conclusiones obtenidas.

2 Motivación

La mayoría de lenguajes de programación utilizados para programar computadores cuánticos hacen uso de circuitos para manejarlos. Esto conlleva a una abstracción muy baja y poco intuitiva. Por ello, se están comenzando a diseñar técnicas de Ingeniería del Software Cuántico [8] que permitirían no solo solucionar problemas relacionados con la programación cuántica, sino cuestiones como el diseño de circuitos [9] y la obtención de requisitos en este tipo de sistemas cuánticos.

Hoy en día, la forma principal con la que interactuar con computadores cuánticos es a través de proveedores que los ofrecen como un servicio más en la nube [10]. Al existir aún pocos computadores disponibles, no todas las peticiones pueden realizarse al instante. Esto puede provocar largas colas de espera para la ejecución del circuito que deseamos y genera incertidumbre respecto a cuándo vamos a obtener el resultado que necesitamos. Es factible realizar una aproximación de este problema a una arquitectura asíncrona, en la cual enviamos información y no sabemos en qué momento obtendremos la respuesta, pero mientras tanto el sistema no está parado, sino que sigue funcionando.

La principal motivación para la invocación de estos servicios utilizando arquitecturas basadas en eventos es alejarse de los paradigmas de comunicación síncronos y así lograr resolver el principal problema anteriormente mencionado: la obtención de los resultados de forma eficiente. AsyncAPI[4] permite generar una arquitectura basada en eventos de forma muy sencilla: mediante un fichero YAML —un contrato— que creará una API asíncrona capaz de reaccionar a envíos y llegadas de mensajes. Debido a la necesidad de una arquitectura asíncrona se ha descartado el uso de OpenAPI[5] ya que esta especificación genera APIs REST síncronas. Desarrollando esta propuesta, es posible unir la computación clásica con la computación cuántica, encapsulando esta última en métodos y funciones clásicos. Estas funciones clásicas invocan código cuántico que se ejecuta en la nube.

3 Generación de servicios cuánticos

3.1 La especificación AsyncAPI

AsyncAPI es una iniciativa bastante reciente basada en OpenAPI. Su principal cometido es el de generar una API asíncrona basada en eventos. Al igual que con

[4] `https://github.com/asyncapi/spec`
[5] `https://github.com/OAI/OpenAPI-Specification`

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

OpenAPI, una de las ventajas principales de utilizar AsyncAPI es la automatización de la generación del código y la obtención de documentación, por lo que reduce de forma considerable el tiempo para obtener la API.

Uno de los problemas que nos podemos encontrar con aplicaciones asíncronas son los protocolos que utilizan; al contrario que con OpenAPI, que genera APIs REST, basadas en HTTP. En los sistemas asíncronos nos encontramos diversos protocolos que funcionan de forma diferente. En el caso de AsyncAPI, se puede utilizar una gran cantidad de protocolos, en los que se incluyen algunos de los más utilizados, como MQTT, AMQP, WebSockets o HTTP.

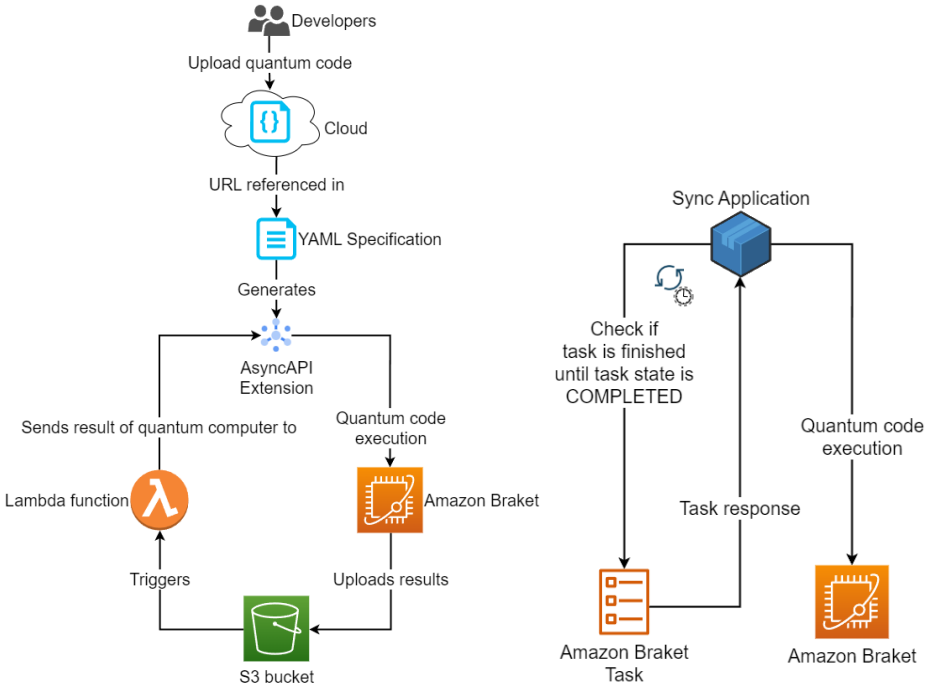
AsyncAPI nos permite definir la API asíncrona teniendo en cuenta los canales de comunicación. Además permite al desarrollador ejecutar cierta lógica de negocio cuando se recibe o manda un mensaje por ciertos canales. La lógica de negocio es específica de cada canal y su ejecución depende de la definición del canal. Si es de tipo *publish*, la lógica se ejecuta cuando la API reciba un mensaje válido sobre ese canal, mientras que si es de tipo *subscribe*, la lógica se ejecuta cuando la API manda un mensaje válido sobre ese canal.

El código generado por AsyncAPI está escrito en JavaScript y es ejecutado por NodeJS. Por tanto,todo lo que los desarrolladores añadan en la lógica de negocio debe estar escrito en JavaScript. Aquí nos encontramos con uno de los primeros problemas: la mayoría del software cuántico está escrito en Python y el código cuántico para Amazon Braket no es una excepción. La solución es simple: invocar el fichero con el código cuántico como un proceso Python desde NodeJS. Una solución alternativa consistiría en utilizar la plantilla de Python para AsyncAPI. No obstante, se ha descartado porque el código generado por esta plantilla puede llegar a contener errores dependiendo del contenido del documento AsyncAPI y solamente funciona con el protocolo MQTT, lo que reduciría las posibilidades de extender el proyecto. Por tanto, se ha elegido el generador de código más estable, que es el generador de NodeJS que utiliza el paquete Hermes, con este generador es posible generar servicios cuánticos en cualquier protocolo que soporte AsyncAPI.

Para la generación de los servicios cuánticos en AsyncAPI, el primer paso consiste en la definición del circuito de Amazon Braket. Este fichero Python es almacenado en la nube y referenciado mediante su URL en el documento YAML que contenga la especificación AsyncAPI. Adicionalmente, es posible indicar en el documento YAML si se quiere crear una función Lambda AWS para completar el ciclo y así poder recibir el resultado del computador cuántico en cuanto se haya completado la operación. Con este contrato, se genera el código de la API asíncrona que es capaz de invocar el servicio cuántico de Amazon Braket.

Como se representa en la Figura 1(a), el código subido por los desarrolladores a la nube se referenciará en la especificación de AsyncAPI extendida. Seguidamente, tras ejecutar el generador de código, se generará una API asíncrona capaz de invocar el código cuántico a Amazon Braket. También creará una función Lambda si así está indicado en el contrato. Cuando se finalice la tarea cuántica, el resultado de esta se almacena en un *bucket* de S3, lo que lanza un evento que invoca la función Lambda. Esta función Lambda creada en la

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.



(a) Arquitectura asíncrona (b) Arquitectura síncrona

Fig. 1. Comparación de generación de servicios cuánticos usando la extensión AsyncAPI 1(a) y utilizando una arquitectura síncrona 1(b)

especificación de AsyncAPI devolverá el valor del computador cuántico al canal que se especifique en el contrato.

3.2 Propuesta de creación de Servicios API Cuánticos

Para la generación del servicio cuántico asíncrono, los desarrolladores han de crear un fichero Python para Amazon Braket y subirlo a un repositorio como GitHub. Deberán obtener la URL de este fichero en formato RAW. Utilizaremos como ejemplo un circuito que genera números aleatorios utilizando una máquina cuántica de Amazon Braket. Para la generación de números aleatorios se utilizan puertas *Hadamard* [11] sobre los *Qubits* correspondientes para que así cambien a un estado de superposición. Dependiendo del número de *Qubits* que sean afectados por estas puertas, se generará un número aleatorio de mayor o menor longitud. En nuestro ejemplo, utilizaremos 5 *Qubits* y sobre cada uno una puerta *Hadamard*. El número aleatorio resultante se encontrará entonces entre los valores 0 y 31.

Con la extensión de AsyncAPI que se ha realizado, se permite añadir un campo adicional sobre un canal. Esta extensión será la URL del fichero Python

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

con el código que se ejecutará como lógica de negocio al recibir/enviar un mensaje por tal canal. La acción que dispara la lógica de negocio depende de la definición del canal en el documento AsyncAPI.

La extensión correspondiente se llama *x-quantum* y presenta dos parámetros: *quantum_ url*, en la cual se debe especificar la URL donde se encuentre el fichero Python correspondiente y *quantum_ machine*, que contiene la máquina en la que se ejecutará. Para que este último campo tenga efecto, el fichero Python debe utilizar los argumentos de línea de comandos al momento de seleccionar la máquina. Se puede ver un ejemplo de uso de la extensión en la Figura 2.

```
channels:
  amazon_braket_operation:
    publish:
      operationId: onOperation
      message:
        $ref: '#/components/messages/operationMsg'
        name: 'operationMsg'
      x-quantum:
        quantum_url: https://raw.githubusercontent.com/jorgecs/pythonscripts/main/randomWithS3.py
        quantum_machine: 'sv1'
```

Fig. 2. Ejemplo de la especificación de la API para el algoritmo de números aleatorios

Además de esto, se ha realizado una extensión adicional que genera una funcion Lambda en AWS. Esta se activa cuando se añade un dato al *bucket* S3 correspondiente. Se genera así una arquitectura que realiza peticiones a una máquina cuántica. Esta envía los resultados a S3, el cual a su vez dispara la función Lambda que envía el dato almacenado en el *bucket* al canal que se haya especificado en el contrato. El resultado puede ser recogido por AsyncAPI u otra aplicación que consuma eventos. Esta segunda extensión se llama *x-quantum-awslambda* y ofrece cuatro parámetros necesarios para la creación de la función Lambda en AWS. Estos parámetros son: *channel*, el canal al que la función Lambda mandará el resultado; *s3_ bucket*, el *bucket* S3 que disparará la función cuando algún elemento se añada; *role_ arn*, el nombre de recurso de Amazon (ARN) del rol de AWS que ejecutará la función; y *region*, la región en la cual se encuentre el S3. La Figura 3 muestra un ejemplo de uso de esta extensión.

```
x-quantum-awslambda:
  channel: 's3_lambda_notification'
  s3_bucket: 'amazon-braket-lambda-bucket'
  role_arn: 'arn:aws:iam::123456789012:role/service-role/s3getobjectrole'
  region: 'us-west-2'
```

Fig. 3. Ejemplo de la especificación de la API para generar una función Lambda en AWS

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

Para la generación de código a partir de estas dos extensiones, se ha modificado el generador de código de AsyncAPI en NodeJS, añadiendo una nueva carpeta llamada *lambda* que contiene los ficheros necesarios para la generación de la función Lambda, que obtienen sus valores con los datos de la plantilla de AsyncAPI. Además se han modificado los *handlers* de la lógica de negocio de los canales. Si además se usa la extensión *x-quantum*, se añadirá el código necesario para obtener el fichero Python y ejecutarlo. Es importante destacar que hasta el momento, la función Lambda generada en AWS solo es compatible con el protocolo MQTT, aunque sigue siendo posible generar los servicios cuánticos con cualquier otro protocolo. Al utilizar la plantilla de NodeJS es posible extenderlo para que pueda funcionar con los demás protocolos disponibles.

Amazon dispone de un servicio de notificación Simple Notification Service (SNS) que envía notificaciones sobre acciones en S3 que posteriormente pueden integrase en modelos asíncronos de publicación y subscripción. Sin embargo, mientras que el servicio SNS proporciona una solución general para enviar notificaciones sobre eventos en AWS, nuestra propuesta ofrece una forma más flexible y personalizada para recibir alertas específicas basadas en criterios definidos por el usuario.

Con todo esto, el desarrollador es capaz de invocar un servicio cuántico dentro de una arquitectura *publish/subscribe*. Además, recibe el resultado del código cuántico de forma asíncrona cuando se encuentre disponible, sin tener que recurrir a peticiones constantes para comprobar en qué momento ha finalizado la operación. La plantilla se encuentra disponible en un repositorio GitHub[6].

4 Ejemplo de uso

Con la ayuda de las dos extensiones creadas para AsyncAPI, se puede crear una arquitectura que permita subsanar los problemas que surgen al trabajar con Amazon Braket. Para comprobar su uso, se puede crear un fichero YAML desde cero que siga la especificación de AsyncAPI o bien, partir de una plantilla de YAML disponible en el siguiente repositorio público[7]. Modificando algunas propiedades como la URL del servidor broker, la URL donde se encuentra el código cuántico y el *bucket* S3 se consigue generar una API asíncrona que cuenta, en este caso, con tres canales. Permite tanto lanzar el código cuántico a Amazon Braket como recuperar el resultado sin tener que realizar peticiones constantes para comprobar el estado de la tarea. Como se indica en el contrato, cuando la API reciba un mensaje en el canal *amazon_ braket_ operation*, se lanzará la ejecución del código cuántico.
Además se genera una función Lambda que mandará los resultados sobre el canal *s3_ lambda_ notification*, los cuales serán recibidos por la API para comprobar de forma más sencilla que el resultado se ha obtenido correctamente.

El proceso a seguir desde que se lanza la aplicación AsyncAPI hasta que se recibe el resultado del computador cuántico es el siguiente: Cuando se ejecuta

[6] `https://github.com/jorgecs/AsyncAPIQuantum`
[7] `https://github.com/jorgecs/AsyncAPISample/blob/main/example.yaml`

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

la aplicación, se crea inicialmente la función Lambda en AWS, preparada para recoger los datos que se añadan al *bucket* S3 y enviarlos. En cuanto un productor mande un mensaje válido sobre el canal *amazon _ braket_ operation*, se recuperará el fichero en la nube que contiene el código cuántico y se ejecutará. Se podrá observar dentro de Amazon Braket que la tarea se ha creado.

Tras pasar el tiempo necesario para que la tarea haya cambiado de estado a *COMPLETED*, su resultado se mandará al *bucket* S3 especificado en el fichero que contiene el código cuántico. Esto activará la función Lambda, siempre y cuando el *bucket* S3 especificado en el contrato y en el fichero que contenga el código Python sean el mismo. Entonces, la función se conectará a la URL del servidor especificada en el contrato y enviará el dato almacenado anteriormente en el *bucket* S3. En este caso, como la propia API intercepta los mensajes que llegan desde el canal *s3_ lambda_ notification*, es posible observar el resultado, otra opción sería mantener consumidores suscritos al canal para que estos reciban los datos y puedan tratarlos de la forma que necesiten.

```
← amazon_braket_operation was received:
{ s3bucket: 'test', s3folder: 'test', machine: 'sv1' }
← s3_lambda_notification was received:
{
  data: '{\n' +
    '    "braketSchemaHeader": {\n' +
    '        "name": "braket.task_result.gate_model_task_result",\n' +
    '        "version": "1"\n' +
    '    },\n' +
    '    "measurements": [\n' +
    '        [\n' +
    '            1,\n' +
    '            0,\n' +
    '            0,\n' +
    '            0,\n' +
    '            1\n' +
    '        ],\n' +
    '        [\n' +
    '            0,\n' +
    '            0,\n' +
    '            1,\n' +
```

Fig. 4. Salida en consola de los mensajes obtenidos en AsyncAPI

Como se puede observar en la Figura 4, inicialmente la API generada por AsyncAPI recibe el mensaje del primer productor, por lo que recupera el fichero con el código cuántico y envía la tarea a Amazon Braket. A continuación, recibe el resultado del computador cuántico en el canal especificado en el contrato, enviado por la Lambda de AWS generada al ejecutar la aplicación. El fichero YAML es un ejemplo que puede cambiarse por completo para dar lugar a arquitecturas diferentes, modificando tanto el tipo como el número de canales o incluso el protocolo del servidor. Se pueden realizar todas las modificaciones pertinentes mientras que siga la especificación establecida por AsyncAPI. También es posible utilizar únicamente la extensión *x-quantum* si no se necesita recoger el resultado y solamente se quiere lanzar el código cuántico a Amazon Braket.

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

Si el servicio simplemente está temporalmente inaccesible o tarda más en responder de lo habitual, se reintenta varias veces antes de dar por fallido el intento y notificar al usuario que hay un problema con el servicio. Además. si se produce un error en los datos recibidos desde Amaozn Braket, se genera una excepción para indicar que no pudo procesar la información correctamente.

5 Trabajos relacionados

Poco a poco están empezando a surgir trabajos que se enfocan en la generación de arquitecturas híbridas, con cierto énfasis en el problema de las colas y cómo minimizarlo. Cada trabajo ofrece una solución diferente utilizando herramientas y métodos diversos.

Trabajos como [12] plantean algo similar a lo propuesto en este artículo. En particular, generan servicios cuánticos desde una API REST síncrona, generada gracias a OpenAPI. Para ello, los autores modifican la especificación OpenAPI para permitir la invocación de servicios cuánticos, encapsulándolos en servicios clásicos. Nuestra motivación es diferente. Utilizamos AsyncAPI para generar una arquitectura que permita, además de invocar los servicios cuánticos, recuperar resultados de forma eficiente sin la necesidad de realizar una gran cantidad de peticiones que pueda llegar a saturar la red, mientras que [12] se centra solo en la creación de una arquitectura híbrida para la invocación de servicios cuánticos. El trabajo [13] también plantea algo similar pero desde una perspectiva diferente. En él se propone la idea de generar aplicaciones híbridas utilizando flujos de trabajo, abordando el problema de las colas para obtener los resultados del computador. Su objetivo es intentar reducir la cantidad de tiempo que el programa se encuentra en cola. Para ello, hacen uso de un entorno de ejecución híbrido con el que han desarrollado un framework para la herramienta de modelado para generar flujos que contengan aplicaciones con código cuántico.

6 Conclusiones

La especificación de servicios cuánticos no es sencilla y las herramientas que existen actualmente no son de gran ayuda ya que son de muy bajo nivel. Aunque hoy en día existen pocas herramientas de ayuda para los servicios asíncronos, con la adaptación de herramientas como AsyncAPI, es posible generar este tipo de aplicaciones de una forma ágil. Con ello, la computación cuántica puede entrelazarse con la computación clásica a través de la servitización. De esta manera se simplifica la generación de una aplicación clásica que utilice código cuántico. La convivencia entre el software clásico y cuántico es posible. En este trabajo se ha realizado una extensión la especificación de AsyncAPI para lograr la invocación de servicios cuánticos, permitiendo integrar software cuántico en las aplicaciones clásicas que lo necesiten. De esta manera, se aboga por la servitización de los servicios cuánticos y clásicos para un desarrollo de ambos utilizando un nivel de abstracción más alto y siguiendo métodos de ingeniería. Todo ello para conseguir una mayor rapidez y menor coste a la hora de crear y desplegar tales servicios.

This work is licensed under a Creative Commons “Attribution-NonCommercial-NoDerivatives 4.0 International” license.

Actualmente, estamos ampliando nuestro trabajo para que la generación de código pueda extenderse a más proveedores de servicios cuánticos—como IBM Quantum—, utilizando más protocolos y abarcando más tipos de aplicaciones.

Agradecimientos Este trabajo forma parte de la ayuda PID2021-1240454OB-C31 financiada por MCIN/AEI /10.13039/50100011033 y por "ERDF A way of making Europe". También está financiado por el proyecto QSALUD project (EXP 00135977 / MIG-20201059) en las líneas de actuación del Centro para el Desarrollo Tecnológico y la Innovación (CDTI); y por el Ministerio de Economía y Transformación Digital del Gobierno de España a través de la convocatoria del proyecto Quantum ENIA - Proyecto Quantum Spain, y por la Unión Europea a través del Plan de Recuperación, Transformación y Resiliencia - NextGenerationEU en el marco de la Agenda España Digital 2025.

Referencias

1. J. Zhao, "Quantum software engineering: Landscapes and horizons," 2020.
2. E. R. MacQuarrie, C. Simon, S. Simmons, and E. Maine, "The emerging commercial landscape of quantum computing," *Nature Reviews Physics*, vol. 2, no. 11, pp. 596–598, 2020.
3. J. M. Garcia-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, and J. M. Murillo, "Quantum software as a service through a quantum api gateway," *IEEE Internet Computing*, 2021.
4. M. Piattini, M. Serrano, R. Perez-Castillo, G. Petersen, and J. L. Hevia, "Toward a quantum software engineering," *IT Professional*, vol. 23, no. 1, pp. 62–66, 2021.
5. H. Singh and A. Sachdev, "The quantum way of cloud computing," in *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, 2014, pp. 397–400.
6. B. Sodhi, "Quality attributes on quantum computing platforms," *arXiv preprint arXiv:1803.07407*, 2018.
7. M. Ripon and M. M. Islam, "An overview on quantum computing as a service (qcaas): Probability or possibility," *International Journal of Mathematical Sciences and Computing*, vol. 2, pp. 16–22, 01 2016.
8. S. Ali, T. Yue, and R. Abreu, "When software engineering meets quantum computing," *Communications of the ACM*, vol. 65, no. 4, pp. 84–88, 2022.
9. R. Pérez-Castillo, M. A. Serrano, and M. Piattini, "Software modernization to embrace quantum technology," *Advances in Engineering Software*, vol. 151, 2021.
10. F. Leymann, J. Barzen, M. Falkenthal, D. Vietz, B. Weder, and K. Wild, "Quantum in the cloud: application potentials and research opportunities," *arXiv preprint arXiv:2003.06256*, 2020.
11. D. J. Shepherd, "On the role of hadamard gates in quantum circuits," *Quantum Information Processing*, vol. 5, pp. 161–177, 2006.
12. J. Romero-Álvarez, J. Alvarado-Valiente, E. Moguel, J. García-Alonso, and J. M. Murillo, "Generación de servicios cuánticos ampliando la especificación openapi," *Sistedes*, 2022.
13. B. Weder, J. Barzen, M. Beisel, and F. Leymann, "Provenance-preserving analysis and rewrite of quantum workflows for hybrid quantum algorithms," *SN Computer Science*, vol. 4, no. 3, p. 233, 2023.

This work is licensed under a Creative Commons "Attribution-NonCommercial-NoDerivatives 4.0 International" license.