

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

**Sistema de Evaluación de Coste para Aplicaciones Híbridas  
(Cuánticas-Clásicas)**

**Cost Evaluation System for Hybrid Applications  
(Quantum-Classical)**

Realizado por  
**Álvaro Manuel Aparicio Morales**

Tutorizado por  
**Javier Cámara Moreno**

y  
**José Manuel García Alonso**

Departamento  
**Lenguaje y Ciencias de la Computación**

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE 2024

# Sistema de Evaluación de Coste para Aplicaciones Híbridas (Cuánticas-Clásicas)

Álvaro M. Aparicio-Morales<sup>1</sup>[0009–0009–5161–5498]

<sup>1</sup> Universidad de Málaga, Málaga, España

<sup>2</sup> Universidad de Extremadura, Cáceres, España

**Abstract.** Due to advancements in quantum computing, its incorporation into current computing systems for resolving complex problems has become increasingly relevant. This implies the coexistence of both computing models, forming what is known as (quantum-classical) hybrid systems. One way to achieve that is through Service-Oriented Computing. These systems would consist of both classical and quantum services, and just as in the classical services applications where finding an optimal deployment configuration is necessary, the deployment of these quantum-classical applications also requires optimization. In classical Service-Oriented Computing exist several tools and methodologies that calculate this optimal distribution of applications. However, there is a lack of tools and methodologies to achieve the optimal distribution for quantum-classical applications. Given this context, this work presents an evaluation system that implements a methodology to address the optimal cost distribution of such hybrid applications. This platform draws upon model-checking techniques and utility theory to obtain an optimal deployment in terms of economic cost. To fulfill that, a workflow starting from an application specification, provides an optimal distribution of the services that comprise it is followed. This evaluation system aims to introduce the use of techniques from classical computing to address the challenge of achieving optimal deployment of hybrid applications, contributing to the advancement of quantum-classical systems and applications development.

**Keywords:** Quantum Software Engineering · Service-Oriented Computing · Probabilistic Model Checking

**Resumen.** Gracias a los avances en computación cuántica, su incorporación en los sistemas de computación actuales para resolver problemas complejos se ha vuelto cada vez más relevante. Esto implica la coexistencia de ambos modelos de computación, formando lo que se conoce como sistemas híbridos (cuántico-clásicos). Una forma de lograrlo es a través de la Computación Orientada a Servicios. Estos sistemas constan de servicios tanto clásicos como cuánticos, y al igual que en las aplicaciones de servicios clásicos donde es necesario encontrar una configuración de despliegue óptima, el despliegue de estas aplicaciones cuántico-clásicas también requiere de ello. En la Computación Orientada a Servicios clásica existen varias herramientas y metodologías que calculan esta distribución

óptima de aplicaciones. Sin embargo, existe una falta de herramientas y metodologías para lograr la distribución óptima en aplicaciones cuántico-clásicas. En este contexto, este trabajo presenta un sistema de evaluación que implementa una metodología para abordar la distribución óptima en coste de dichas aplicaciones híbridas. Esta plataforma se basa en técnicas de verificación de modelos y teoría de utilidad para obtener un despliegue óptimo en términos de coste económico. Para lograrlo, se sigue un flujo de trabajo que parte de una especificación de aplicación, proporcionando una distribución óptima de los servicios que la componen. Este sistema de evaluación tiene como objetivo introducir el uso de técnicas de computación clásica para abordar el desafío de lograr un despliegue óptimo de aplicaciones híbridas, contribuyendo al avance de los sistemas y desarrollo de aplicaciones cuántico-clásicas.

**Keywords:** Ingeniería de Software Cuántico · Computación Orientada a Servicios · Model Checking Probabilístico.

**Tutores:** Javier Cámara<sup>1</sup>[0000-0001-6717-4775] y Jose Garcia-Alonso<sup>2</sup>[0000-0002-6819-0299]

## 1 Introducción

Hoy en día, en el ámbito empresarial, existen un conjunto de sistemas informáticos que son altamente complejos y que se encargan de tareas de enorme cantidad de procesamiento para resolver problemas en tiempo real. Sin embargo, en diversas áreas de esta rama tecnológica, existen desafíos cuya resolución resulta compleja, prolongada y, en algunos casos, inalcanzable para la actual forma de computación [1]. En particular, en el ámbito de la Ciencia e Ingeniería de Datos, el procesamiento de grandes volúmenes de información es una tarea que puede consumir largos periodos de tiempo, como ocurre en el entrenamiento de modelos de inteligencia artificial o en el análisis en tiempo real de grandes conjuntos de datos provenientes de sensores. Con la tendencia creciente en la generación de enormes volúmenes de datos, motivada entre otros factores por el uso masivo de sensores en ciudades inteligentes (smart cities) e industrias conectadas (Industria 4.0), así como por el desarrollo de tecnologías emergentes como los gemelos digitales, se plantea necesaria la exploración de nuevas soluciones y paradigmas tecnológicos para abordar el análisis de estos datos, como por ejemplo la exploración de otras vías de realizar computación, no a priori como sustitución sino como complemento para mejorar las actuales capacidades computacionales. Una de estas alternativas, es la computación cuántica.

De este nuevo paradigma computacional cuántico se espera una alta capacidad de cómputo frente a los vigentes computadores. Esta alta capacidad tiene su origen en el aprovechamiento de las propiedades de la física cuántica para hacer computación. Este modelo, está basado en la utilización de los bits cuánticos o *cubits*, la unidad mínima de información cuántica, en lugar de los bits clásicos en los computadores tradicionales. Los cubits poseen las propiedades cuánticas de superposición, y entrelazamiento que les permiten encontrarse en un estado

indeterminado hasta su medición, además de estar relacionados con otros cubits de modo que su estado no puede describirse de forma independiente al estado de los demás. Gracias a estas propiedades, los computadores cuánticos tienen la capacidad computacional suficiente para abordar ciertos problemas, donde el método de computación actual queda limitado [2]. Esta habilidad computacional está fundamentada en la solución de problemas cuánticos de tiempo polinómico con cierto margen de error, conocidos como BQP. Se presume que BQP incluye a P, y que además existen ciertos problemas dentro de la clase BQP que están más allá de P. Estos problemas son los que despiertan un interés especial, tanto por su aplicabilidad a problemas ya conocidos, como por la demostración de la supremacía cuántica. Esta última sostiene que una computadora cuántica puede resolver un problema que ninguna computadora clásica puede solucionar en un tiempo razonable.

De entre los problemas conocidos que pueden ser resueltos por una computadora cuántica en tiempo razonable, destacan la descomposición de números enteros con el uso del algoritmo de Shor [3] o la búsqueda en una secuencia no ordenada de datos con N componentes mediante el algoritmo de Grover [4]. Estos algoritmos son llamativos por su ámbito de aplicación en Ciberseguridad y en la Ciencia de Datos respectivamente. Otros posibles campos de aplicación destacados pueden ser el ámbito climático contribuyendo a la realización de cálculos meteorológicos más precisos [5], el sector de la biomedicina mediante la generación de tratamientos farmacológicos personalizados [6] o la industria con la optimización en los procesos de producción. Todas estas áreas, requieren de soluciones donde se procesan no sólo grandes cantidades de datos, sino también ciertos cálculos complejos donde los computadores convencionales quedan limitados.

Por tanto, dada su ventaja, resulta interesante incorporar estos ordenadores con los sistemas de cómputo actuales para realizar tareas donde se requieran de altas capacidades de computación. Esta coexistencia de ambos modelos es lo que se denomina sistema híbrido (clásico-cuántico). La construcción de estos sistemas híbridos puede enfocarse desde la computación orientada a servicios permitiendo aprovechar las ventajas que ofrece la servitización de procesos como por ejemplo el desacoplamiento, la flexibilidad o el escalado entre otras, por lo que resultaría sencillo su incorporación en la fase de tratamiento en el ciclo de vida del dato [7]. Sin embargo, dado el estado actual de la ingeniería de servicios en el ámbito de la computación cuántica, aún no resulta posible el desarrollo de los servicios cuánticos de la misma forma que se desarrollan los servicios convencionales [8]. Además, debido al estado actual de la tecnología para la construcción de un computador cuántico, éstos, solamente pueden estar alojados en grandes infraestructuras como pueden ser los centros de datos o edificios de investigación de grandes multinacionales [9].

No obstante, a través del modelo llamado Quantum Computing as a Service (QCaas) resulta posible acceder a los computadores cuánticos mediante los proveedores de servicios en la nube haciendo posible la incorporación de la computación cuántica en las aplicaciones de servicios usando los kits de desarrollo

software (SDK) elaborados por cada proveedor. Con esta alternativa resulta posible la generación de las denominadas aplicaciones híbridas (clásico-cuánticas). Este tipo de aplicaciones están conformadas por servicios clásicos y servicios híbridos. Un servicio híbrido, se define como un servicio clásico donde en una de sus rutas tiene implementado un algoritmo cuántico [10]. Esto implica, que cuando se realice una llamada a esa ruta del servicio híbrido, se ejecutará un algoritmo cuántico. Por tanto, en las configuraciones de las aplicaciones híbridas (clásico-cuánticas), tendremos por un lado un conjunto de servicios puramente clásicos y otros servicios clásicos en cuyo interior hay configurada una ruta donde se configura un algoritmo cuántico para ser lanzado en los proveedores de servicios en la nube. Estas configuraciones de aplicaciones híbridas, resultan ser los primeros pasos en el desarrollo del área de estudio de la Computación Orientada a Servicios Cuánticos dentro del dominio de la Ingeniería de Software Cuántico.

Como se recoge en el *Manifiesto de Talavera* [2] existe cada vez más necesidad del desarrollo de soluciones cuánticas motivadas por la multitud de aplicaciones en diferentes ámbitos, y al igual que ha sucedido para el software clásico, es necesaria la creación y/o adaptación de técnicas y procesos que permitan a los ingenieros de software la creación y desarrollo de soluciones híbridas (clásico-cuánticas) que aprovechen al máximo la capacidad computacional de esta nueva generación de sistemas. Por ello, será necesario enfrentarse a nuevos y/o similares problemas a los que ya se han abordado para el desarrollo de aplicaciones de servicios clásicas [11] pero esta vez, contemplando las características intrínsecas de la computación cuántica y servicios híbridos.

Entre los retos actuales a los que se enfrenta la Computación Orientada a Servicios Cuánticos se encuentra la gestión de la demanda y capacidad [12] de los flujos de trabajos híbridos. Dentro de este reto podemos encontrar por un lado el coste de la ejecución de un algoritmo cuántico, que motivado por la novedad de esta tecnología y por el modelo de pago por uso que habita en los entornos de servicios en la Nube, repercute en elevadas cuantías económicas en la ejecución de estos algoritmos. Por otro lado, existe la problemática en la algoritmia cuántica de la selección de un computador cuántico que sea adecuado para un algoritmo o conjunto de algoritmos [13]. Además de este coste, hay que considerar el precio del resto de los servicios que conforman la aplicación. Asimismo, las aplicaciones de servicios, tienen que ser desplegadas y para la obtención de ese despliegue, actualmente, existen diversas herramientas que resuelven este problema, denominado *Placement Problem*. Estas herramientas, tienen en cuenta ciertas propiedades como el consumo energético, el rendimiento de las alternativas de despliegue o la calidad del servicio entre otras [14, 15] para la obtención de una distribución óptima de los servicios. En la actualidad, disponemos de diversas herramientas y técnicas que facilitan la evaluación de ciertas propiedades como las mencionadas anteriormente, contemplando las diferentes alternativas de distribución en presencia de niveles de incertidumbre. Esta incertidumbre puede surgir, por ejemplo, de la falta de control sobre los servicios proporcionados por terceros, o de las latencias de red que son difíciles de prever. Para lograr la evaluación, se emplean técnicas de verificación formal cuantitativa, como el model

checking probabilístico, cuya eficacia ha sido probada en la evaluación de arquitecturas alternativas en presencia de incertidumbre en diversos contextos [14]. Sin embargo, existe ausencia de técnicas y procesos que aborden el despliegue óptimo contemplando los servicios híbridos.

Con la idea de suplir esta carencia, en este trabajo, se presenta un sistema evaluador que resuelve el *placement problem* con la elección de una arquitectura de despliegue óptima en coste y rendimiento para las aplicaciones híbridas (clásico-cuánticas) de servicios aplicando técnicas de model checking y usando la teoría de la utilidad. Esta búsqueda de una arquitectura de distribución óptima se convierte en un problema multidimensional, que comienza con la selección de un computador cuántico adecuado para un determinado algoritmo, continuando con la elección de una distribución equilibrada en coste y rendimiento. Adicionalmente, para comprobar y evaluar el sistema implementado se ha realizado el desarrollo de una prueba de concepto para obtener la distribución óptima de una aplicación híbrida de servicios.

El resto de la memoria sigue la siguiente estructura. La Sección 2 expone los antecedentes del presente trabajo. La Sección 3 describe la problemática que aborda este trabajo. La Sección 4 presenta los detalles de la propuesta abordando la metodología, diseño, implementación y evaluación de la propuesta así como trabajos relacionados. En la Sección 5 se expondrán las conclusiones y trabajos futuros.

## 2 Antecedentes

### 2.1 Teoría de utilidad y Eficiencia de Pareto

La teoría de la utilidad, es una hipótesis que tiene su origen en el ámbito económico. Esta teoría permite expresar la toma de decisiones de los usuarios según el grado de utilidad o satisfacción que obtienen en la adquisición de un artículo determinado [16]. Fue formulada por varios economistas, entre los que destacan David Ricardo y Adam Smith autor del libro "*La riqueza de las naciones*" [17], libro influyente en economía donde se expone la idea de que la riqueza de una nación no reside en la acumulación de riquezas sino en las propiedades de producción y comercio de sus ciudadanos.

En este sentido, la teoría de la utilidad puede extenderse a otras disciplinas donde se requiera una toma de decisiones que impliquen propiedades [18]. En el contexto de los sistemas distribuidos, la teoría de la utilidad puede ofrecerse como un recurso fundamental para la selección de una arquitectura de despliegue de las aplicaciones de servicios. Estas aplicaciones pueden llegar a ser sistemas complejos que pueden estar compuestos por muchos servicios independientes que cooperan entre sí por lo que la elección de cómo y dónde ubicar estos servicios puede repercutir significativamente en el rendimiento, escalabilidad y fiabilidad del sistema. Con la teoría de la utilidad, es posible considerar múltiples factores al mismo tiempo. En esta propuesta, se pretende hacer uso de las funciones de utilidad de tipo *weighted sum* (suma ponderada). Este tipo de función, permite

capturar un conjunto de preferencias del usuario sobre distintas dimensiones en un espacio multiobjetivo, pudiéndoles asignar un peso. Con esta idea, las decisiones de diseño de la distribución pueden optimizarse tomando las preferencias de usuario, evaluando y comparando diferentes configuraciones de despliegue de los servicios alcanzando un valor de utilidad para cada una de ellas.

Al tratarse de un problema multiobjetivo, no es posible encontrar una solución única que optimice varios objetivos ya que a menudo pueden resultar conflictivos entre sí. Este es el caso de los objetivos de rendimiento y coste, donde no es posible encontrar una única solución donde el rendimiento sea máximo y el coste sea mínimo ya que un mayor rendimiento repercute directamente en un mayor coste. Por tanto, lo que buscamos en la optimización de problemas multiobjetivo es su espacio de soluciones de Pareto.

El espacio de soluciones de Pareto es un conjunto de todas las soluciones factibles que no pueden ser mejoradas en un objetivo sin empeorar al menos uno de los otros objetivos. Este concepto es una manifestación de la eficiencia de Pareto, que sostiene que una solución es eficiente si no se puede mejorar un objetivo sin empeorar otro [19]. Al explorar este espacio de soluciones, será factible seleccionar la solución que mejor se adapte a las preferencias del usuario. Esta solución seleccionada será aquella que maximice la eficiencia de Pareto y proporcione un equilibrio entre los objetivos en conflicto.

## 2.2 Model Checking Probabilístico

El *model checking* [20] es un método automático para la verificación formal de un sistema. Este método utiliza lógica temporal para especificar propiedades sobre el comportamiento de los sistemas a lo largo del tiempo. Entre las lógicas temporales, algunos ejemplos populares incluyen la conocida como Computation Tree Logic (CTL) [21], o Linear Temporal Logic (LTL) [22]. En esta técnica se modela, tanto el sistema, como las propiedades a verificar utilizando lenguajes de especificación formal. Sin embargo, este método fue inicialmente diseñado para la verificación de propiedades cualitativas de un sistema, es decir, si éstos sistemas, en su evolución, cumplen o no un conjunto de propiedades definidas. No obstante, en muchos sistemas es necesario verificar los denominados *aspectos cuantitativos de la corrección* [23]. Para gestionar propiedades cuantitativas de los sistemas como por ejemplo el tiempo de respuesta o consumo energético, así como para verificar sistemas cuya evolución está gobernada por procesos estocásticos, fue necesaria su extensión [20].

Para este cometido, se desarrolló el *model checking probabilístico* [24] que no solo extiende las lógicas temporales clásica a variantes probabilísticas, como la Computation Tree Logic Probabilística (P-CTL), sino que también introduce nuevos modelos matemáticos para describir sistemas estocásticos, como las Cadenas discretas de Markov (DTMCs, Discrete-Time Markov chains) y las Cadenas de Markov Continuas (CTMCs). Además, los algoritmos de verificación también se adaptan para manejar la evaluación de propiedades cuantitativas probabilísticas en estos nuevos modelos, como por ejemplo el consumo de energía, rendimiento o coste, entre otras.

En el *model checking probabilístico*, cuando la evolución del sistema está descrita por probabilidades discretas, el tipo de modelo utilizado es el que hace uso de las cadenas discretas de Markov. Las DTMC, por sus siglas en inglés, permiten, en los sistemas con transiciones probabilísticas, capturar su comportamiento estocástico a lo largo del tiempo y así poder garantizar ciertas propiedades cuantitativas sobre el sistema. Cuando en la evolución del sistema, además se incluyen toma de decisiones, se utiliza el análisis cuantitativo de los Procesos de Decisión de Markov, que consiste en calcular la recompensa máxima o mínima de una determinada propiedad del sistema. Esto permite ampliar la verificación de las propiedades del sistema contemplando, además, la optimización de decisiones a lo largo de sus transiciones de estado [20].

La especificación de las propiedades de estos sistemas puede incluir aspectos como la probabilidad de fallo de un componente o la esperanza de tiempo hasta alcanzar un estado objetivo. Una vez definidas las propiedades, se emplea un algoritmo de *model checking* para explorar el conjunto de estados del modelo y verificar si las especificaciones se cumplen en el sistema.

En este sentido, el uso del *model checking probabilístico* es clave para garantizar la fiabilidad y seguridad en sistemas que operan bajo condiciones de incertidumbre, y su aplicación es vital en la verificación de sistemas críticos, como centrales nucleares o sistemas de aviación [25]. Además del *model checking probabilístico*, existen otros algoritmos de *model checking* que se adaptan a las características del sistema y al tipo de propiedad que se quiera verificar [26]. La elección del algoritmo adecuado dependerá del contexto específico del sistema en análisis.

### 2.3 Computación Cuántica

El concepto de computación cuántica puede remontarse al año 1981, cuando el célebre físico teórico Richard Feynman planteó la necesidad de tener un computador que permitiese la simulación de sistemas físicos a través de la siguiente cita:

*“Nature isn’t classical, dammit, and if you want to make a simulation of nature, you’d better make it quantum mechanical, and by golly, it’s a wonderful problem because it doesn’t look so easy.”* [27]

**Traducción:** *“La naturaleza no es clásica, maldita sea, y si tu quieres hacer una simulación de la naturaleza, más te vale que sea mediante la mecánica cuántica, y por Dios, es un problema maravilloso porque no parece fácil.”*

A partir de este momento se inició el viaje hacia el desarrollo de una de las tecnologías más prometedoras para la revolución de la computación [28].

Tras las declaraciones de Feynman, comenzaron a surgir diversos artículos en los que, de forma teórica, se demostraba la ventaja de los computadores cuánticos frente a los computadores convencionales en la resolución de ciertos



tipos de problemas. Sin embargo, no fue hasta la presentación del algoritmo de Shor elaborado por Peter Shor y el algoritmo de Grover realizado por Lov Grover que sirvieron como punto de inflexión para reforzar la investigación sobre algoritmos cuánticos y su posible utilidad para la resolución de problemas gracias a la aceleración cuántica [29].

Por un lado, el algoritmo de Shor demostraba la capacidad para factorizar números enteros en tiempo polinómico, algo que se considera imposible para un computador convencional y que es la base de los protocolos de criptografía actuales. Por otro lado, el algoritmo de Grover, presenta una aceleración cuadrática en la resolución del problema de búsqueda de un elemento en una secuencia desordenada de  $N$  elementos, siendo aplicable para la búsqueda en grandes bases de datos dentro del área de la Ciencia de Datos [30]. Esto sin duda resultó ser el precedente del concepto de *supremacía cuántica*.

La supremacía cuántica es un concepto que determina que habrá algún momento en el tiempo, cuando la tecnología de computación cuántica tenga mayor madurez, en el que estos computadores cuánticos superarán la capacidad de los actuales computadores resolviendo en tiempo factible, un problema que no puede ser abordado por un ordenador convencional [31]. Este concepto de supremacía cuántica quedó demostrado por la empresa Google en el artículo llamado “*Quantum supremacy using a programmable superconducting processor*” [31] y en el trabajo realizado por un grupo de investigadores titulado “*Strong quantum computational advantage using a superconducting quantum processor*” [32].

Por tanto, su aplicabilidad en diversos sectores resulta imprescindible para poder aprovechar su alta capacidad de cómputo para abordar tareas que son dificultosas para un computador clásico. Ejemplos de esto se reflejan en el sector sanitario, con la generación de fármacos especializados para cada paciente, descubrimiento de nuevos materiales en el sector industrial o modelado preciso de sistemas meteorológicos [2].

Para entender cómo este tipo de computadores puede tener esta alta capacidad computacional, es necesario entender algunos conceptos básicos.

Un computador cuántico puede definirse como un dispositivo que usa las propiedades de la mecánica cuántica para hacer computación. Estas propiedades son la superposición y el entrelazamiento cuántico.

- *Superposición*: La superposición consiste en la capacidad de una partícula de encontrarse en múltiples estados simultáneamente. Permitiendo conocer su estado cuando se realiza la medición de la partícula. A este proceso de medición se le llama colapso del sistema.
- *Entrelazamiento*: El entrelazamiento es un caso especial de superposición en el que dos o más partículas están correlacionadas de forma que el estado de cualquiera de ellas, no puede definirse de forma independiente del estado de las otras incluso si están físicamente separadas.

Concretamente, un computador cuántico está formado por un conjunto de partículas denominadas cubits que son la unidad mínima de información cuántica. Estos cubits, poseen las propiedades mencionadas anteriormente y gracias a ellas, tienen esta alta capacidad computacional.

Un cúbit, es una combinación lineal de dos estados base  $|0\rangle$  y  $|1\rangle$ . Matemáticamente, a través de la notación de Dirac<sup>3</sup> el estado de un cúbit, puede representarse de la siguiente manera:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Los coeficientes de amplitud  $\alpha$  y  $\beta$  son números complejos que representan las amplitudes de probabilidad de los estados cuánticos. En un computador cuántico, los coeficientes asociados a los estados de los cubits van fluctuando según se vayan aplicando los operadores cuánticos.

Según el postulado de la extensión cuántica de la tesis Church-Turing [33]:

*“Cualquier dispositivo de computación física puede ser simulado de forma eficiente por un computador cuántico tolerante a fallos.”*

Esto implica que la computación clásica, es un caso particular de la computación cuántica ya que las operaciones clásicas otorgan valores discretos, 0 ó 1, a los coeficientes de amplitud, haciendo que siempre se cumpla que  $\alpha = 0$ , entonces el cúbit se encuentra en un estado determinado con valor uno, o que  $\beta = 0$ , lo que implica que el estado del cúbit es cero<sup>4</sup>.

Actualmente, los computadores cuánticos no son tolerantes a fallos porque son muy sensibles al ruido motivado por la naturaleza cuántica de los cubits y su interacción con el medio [34], e influyen en la aparición de imprecisiones en los cálculos. Además, se encuentran en una etapa en la que no son lo suficientemente grandes o estables para la corrección de estos errores. A esta etapa se le conoce como Noisy Intermediate-Scale Quantum Computers (NISQ-Computers). Otro de los problemas que se tiene que hacer frente en el desarrollo de estos computadores, es lo que se conoce como **tiempo de coherencia**. Este concepto es bastante importante y se refiere al periodo en el que un sistema cuántico está estable en estado de superposición, es decir, que su información cuántica es accesible. Debido al estado actual del desarrollo tecnológico del hardware cuántico, los tiempos de coherencia no son lo suficientemente amplios permitiendo que el cúbit colapse por interferencias externas no deseadas. A este colapso se le conoce como **decoherencia**.

Aunque estos computadores cometan errores, existen varios estudios como los trabajos de Younseok Kim et al. [35] y McGeoch, Catherine C et Farré, Paul [36] que demuestran la posibilidad de desarrollar soluciones usando los actuales computadores NISQ. Esto es lo que se conoce como *Quantum Utility*.

Estos actuales modelos de computadores NISQ pueden clasificarse en tres tipos [1]:

- *OneWay*: Este modelo prepara un estado cuántico altamente entrelazado al que se le van realizando mediciones.

<sup>3</sup> La notación de Dirac, también conocida como notación bra-ket, es una forma común de representar vectores cuánticos en la mecánica cuántica.

<sup>4</sup> Esto es demostrable usando la ley de Born. [33]

- *Adiabatic*: Basado en la computación adiabática, se inicia el estado cuántico a través de los coeficientes de la función Hamiltoniana que se desea minimizar. Tras esta iniciación, el sistema evoluciona lentamente hasta un estado que contiene la solución mínima del Hamiltoniano. Este tipo de computadores son usados actualmente para la búsqueda de la solución a problemas de optimización.
- *Gate-Based*: Este modelo de computador, está basado en el uso de puertas cuánticas para operar el estado de los qubits.

Este último modelo, *gate-based*, es el más común dentro de los computadores cuánticos disponibles y es el que más flexibilidad ofrece para la implementación de algoritmos cuánticos, además de ser el que más se asemeja a los conceptos de circuito lógico de una computadora clásica. Por estos motivos, este modelo de computador será el que se tome como referencia a lo largo del trabajo.

Tal y como se ha mencionado anteriormente, los operadores del modelo *gate-based* son puertas lógicas al igual que ocurre en el modelo clásico, pero estas puertas operan con qubits en vez de con bits y se denominan puertas lógicas cuánticas. Este tipo de puertas, tiene una característica especial, y es que son puertas reversibles [33], es decir, que cualquier transformación realizada en un qubit o un sistema de qubits mediante una puerta cuántica puede ser deshecha o invertida mediante una operación cuántica que lleva el sistema de vuelta a su estado original. Sin embargo, la reversibilidad de puertas es posible, sólo si el sistema cuántico es ideal, y dado el actual estado de la computación cuántica, esta característica de reversibilidad puede verse afectada debido a la existencia de los ya mencionados errores cuánticos y los tiempos de decoherencia.

A diferencia de los computadores clásicos y las comúnmente conocidas puertas AND, OR, XOR y NAND, este tipo de puertas no son reversibles, por lo que no pueden ser aplicadas en un computador cuántico. Por tanto, del conjunto de puertas lógicas primarias de la computación convencional, solamente puede ser utilizada la puerta NOT ya que sí es reversible. En la Figura 1, se han destacado tres puertas cuánticas, *Hadamard*, *Swap*, y *Fase*.

Cuando se implementa una puerta cuántica a un qubit, el resultado, sin error, se obtiene mediante la multiplicación matricial entre la matriz correspondiente a la puerta aplicada y el vector del estado del qubit. Por tanto, para cada puerta lógica cuántica existe una matriz que contiene la operación de transformación sobre el qubit, como puede observarse en la Figura 1.

En este contexto, un algoritmo cuántico se implementa mediante la construcción de un circuito cuántico a través de la concatenación de diferentes puertas cuánticas que operan sobre los qubits del sistema. Actualmente los diferentes proveedores de computadores cuánticos, ofrecen los Kits de Desarrollo Software (SDK) que haciendo uso de los entornos de programación y lenguajes de alto nivel actuales, permiten la creación de circuitos cuánticos mediante las llamadas a las librerías específicas de cada proveedor. En las Figuras 2 y 3 puede observarse un ejemplo de circuito cuántico desarrollado en el entorno IBM Quantum Composer <sup>5</sup>.

<sup>5</sup> <https://quantum.ibm.com/composer>

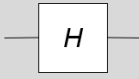


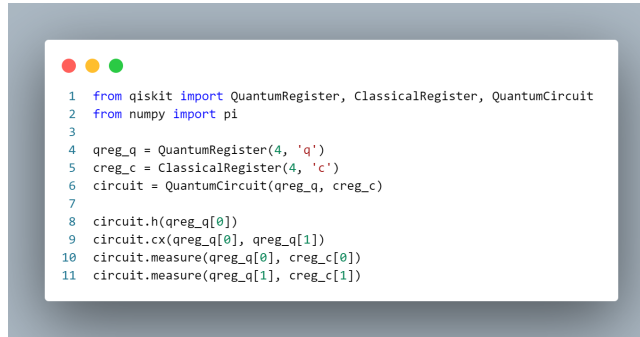
Nombre	Descripción	Matriz de Operaciones	Imagen
<b>HADAMARD</b>	Transforma los estados base en superposiciones	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	
<b>SWAP</b>	Intercambia dos cúbits	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
<b>FASE</b>	Añade una fase a un estado cuántico	$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$	

Fig. 1. Ejemplos de Puertas Cuánticas

Actualmente, el desarrollo de algoritmos cuánticos se realiza a través de lo que se denomina un circuito cuántico, en el que un conjunto de puertas cuánticas opera los cubits del sistema. Estos circuitos cuánticos tienen tres aspectos principales:

- Profundidad o *Depth*: Número máximo de puertas cuánticas aplicadas secuencialmente en la línea de vida de un cúbit del circuito.
- Anchura o *Width*: Es el número de cubits utilizados en el circuito cuántico.
- Ejecuciones o *Shots*: Número de veces que el circuito cuántico va a ser lanzado en la máquina cuántica.

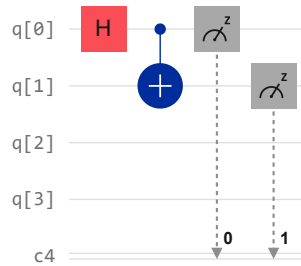
Para la ejecución del circuito cuántico desarrollado, al igual que ocurre en computación clásica, éste debe de ser sometido a un proceso de “compilación” que, en este contexto, se denomina transpilación. De forma resumida, este proceso consiste en traducir el circuito cuántico desarrollado a un nuevo circuito que utiliza únicamente el conjunto de puertas cuánticas soportadas por la máquina cuántica en la que se vaya a ejecutar el circuito original. Esta tarea, es realizada por el transpilador de la máquina cuántica y debido a la falta de estandarización en el hardware cuántico, no todos los computadores soportan el mismo tipo de puertas cuánticas, por lo que el circuito resultante de la transpilación, puede diferir de un proveedor a otro. Además, durante este proceso, se llevan a cabo tareas de optimización para reducir el número de puertas utilizadas. Finalmente, en la última etapa del proceso, se realiza la asignación de los cubits lógicos del circuito transpilado a los cubits físicos del computador y se procede a la ejecución.



```

1 from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
2 from numpy import pi
3
4 qreg_q = QuantumRegister(4, 'q')
5 creg_c = ClassicalRegister(4, 'c')
6 circuit = QuantumCircuit(qreg_q, creg_c)
7
8 circuit.h(qreg_q[0])
9 circuit.cx(qreg_q[0], qreg_q[1])
10 circuit.measure(qreg_q[0], creg_c[0])
11 circuit.measure(qreg_q[1], creg_c[1])

```

**Fig. 2.** Código Circuito Cuántico de Bell**Fig. 3.** Visual de Circuito Cuántico de Bell

Como puede observarse, el proceso de desarrollo y ejecución de algoritmos cuánticos es aún de bastante bajo nivel por lo que resulta necesario avanzar en el desarrollo de herramientas que permitan elaborar soluciones computacionales a alto nivel. Tal y como se recoge el artículo de Zhao [37], están comenzando a aparecer trabajos que abordan el desarrollo de lenguajes de programación que puedan facilitar la implementación de algoritmos cuánticos. Sin embargo, aún queda mucho camino por recorrer en el desarrollo de técnicas, metodologías y herramientas a lo largo de todo el ciclo de vida del software cuántico, que nos permitan elaborar soluciones de la misma forma que lo hacemos en el mundo clásico. Y esto, solamente puede conseguirse si se realiza desde el punto de vista de la Ingeniería de Software [12].

### 3 Naturaleza del Problema

La computación cuántica está llamada a ser una tecnología revolucionaria que nos permita abordar problemas computacionales que, actualmente, no son posibles de afrontar [2]. Es por ello, por lo que la comunidad científica, la industria

y sector público, están trabajando y colaborando de forma conjunta en el desarrollo de todas las áreas implicadas para que esta novedosa tecnología pueda ser usada [38]. Esto implica que se tiene que avanzar en aspectos de telecomunicaciones, hardware, materiales, así como del software, tanto en el desarrollo de algoritmos como en la creación y/o adaptación de procesos y herramientas ya utilizadas.

Sin embargo, para alcanzar este objetivo, desde el ámbito informático, es necesario proceder del mismo modo que sucedió con el software clásico. Eso significa que es necesario el desarrollo de una ingeniería de software cuántico, que puede definirse como la disciplina que abarca el diseño, desarrollo, pruebas y mantenimiento de software específicamente creado para computadoras cuánticas, utilizando principios y prácticas de la ingeniería de software tradicional adaptadas a la naturaleza de la computación cuántica. Para ello, es necesario realizar una revisión exhaustiva a lo largo del ciclo de vida del software para adaptar y/o crear los métodos y procesos necesarios para lograr soluciones que sean capaces de aprovechar plenamente el potencial de los computadores cuánticos [2].

Resulta lógico pensar que abordar la revisión de la Ingeniería de Software y adaptarla al mundo cuántico, requiere del estudio de los desafíos y problemas que surgen al incorporar el software cuántico en la construcción de nuevas soluciones en cada una de las fases del ciclo de vida de software [39].

Actualmente, el software moderno casi nunca trabaja de forma aislada, por lo que los sistemas actuales normalmente son creados como un conjunto de componentes software globales e interconectados; y es evidente asumir, que el software cuántico siga una pauta similar en el que un nuevo tipo de sistema con componentes clásicos y cuánticos se convierta en el estándar de desarrollo [40]. De hecho, los primeros pasos de la computación cuántica van en la dirección de la forma híbrida [41] donde se pueden observar que algunos algoritmos cuánticos relevantes necesitan ya de una parte clásica, así como que la mayoría de las computadoras cuánticas, están aún gestionadas por sistemas clásicos. Por tanto, los esfuerzos para definir una arquitectura híbrida para la ingeniería de software cuántica son necesarios.

Esta integración de componentes clásicos y cuánticos definiendo una arquitectura híbrida, puede ser abordada desde la perspectiva de la Computación Orientada a Servicios [8]. A través de este área, sería posible abordar la conformación de las arquitecturas híbridas, con el desarrollo de servicios cuánticos. No obstante, dado el contexto actual, no es posible implementar un servicio cuántico de la misma forma que un servicio clásico [9]. Empero, gracias a los SDK de los distintos proveedores de servicios y al enfoque propuesto en el trabajo de Garcia-Alonso et al. [10], este problema, es posible enfocarlo mediante el concepto de servicio híbrido donde una de las rutas de este servicio, contiene la implementación de un algoritmo cuántico. Además, con el modelo QCaaS, los algoritmos cuánticos pueden ser desarrollados a través de las SDKs mediante la mencionada construcción de circuitos cuánticos que pueden ser ejecutados en la nube por un simulador o un computador cuántico real.

De esta forma, mediante el paradigma de la computación orientada a servicios, es posible integrar la utilización de computadoras cuánticas en aplicaciones de servicios. Siendo esta perspectiva considerada como los primeros pasos en el área de la Computación Orientada a Servicios Cuánticos.

A pesar de ello, este enfoque no está exento de desafíos. Actualmente, existen varios estudios que abordan los retos a los que se enfrenta la ingeniería de software cuántico, más concretamente en el área del desarrollo de arquitecturas híbridas y que son recogidos en el trabajo de Murillo et al. [12]. Estos retos son:

- Falta de métodos y herramientas que faciliten las decisiones arquitecturales de las aplicaciones híbridas (clásico-cuánticas).
- Definición de patrones de diseño para sistemas híbridos.
- Colaboración con el entorno industrial para la implantación de sistemas híbridos para detectar de factores relevantes.
- Técnicas para la evolución de las arquitecturas híbridas.

En el trabajo que aquí se presenta, se expone un sistema evaluador que facilite la decisión arquitectural sobre la determinación de un despliegue óptimo de los servicios que componen una aplicación híbrida (clásico-cuántica) haciendo uso de técnicas de model checking probabilístico que permiten estudiar las propiedades cuantitativas de coste y rendimiento de cada posible alternativa de despliegue.

## 4 Sistema de Evaluación de Coste para Aplicaciones Híbridas

En esta sección, abordaremos el diseño, implementación y evaluación del sistema para la obtención de una distribución óptima en coste y rendimiento de aplicaciones híbridas (clásico-cuánticas) así como la metodología llevada a cabo para la elaboración de este trabajo. Tras explicar la metodología llevada a cabo, se aborda el diseño de la metodología que implementa el sistema evaluador, explicando cada uno de sus componentes y la cuestión que resuelven. A continuación, se detalla cómo se ha llevado a cabo el desarrollo de la propuesta comentando los retos técnicos para llevar a cabo su funcionamiento. Posteriormente, en el apartado de evaluación, se define el caso de uso utilizado para la comprobación de la correcta ejecución del flujo implementado. Finalmente, se compara y contrasta este trabajo con otros relacionados.

### 4.1 Metodología

Para cumplir con los objetivos generales y específicos del proyecto, es necesaria una planificación adecuada de las actividades. Dado el enfoque investigativo del proyecto, se ha optado por la metodología Design Science, la cual es ideal para proyectos de investigación, especialmente en campos como la Ingeniería y la Informática. Esta metodología busca dar un enfoque práctico a los objetivos mediante la creación de artefactos útiles, como modelos, métodos o innovaciones tecnológicas. Además, Design Science ve a los investigadores como diseñadores

activos, proporcionando pautas para evaluar e iterar los avances en los proyectos. A lo largo de cuatro meses, se ha seguido un proceso cíclico de descripción del problema (**T1**), diseño (**T2**), implementación (**T3**) y desarrollo de la memoria (**T4**) correspondiente a cada módulo. En el primer mes, se trabajó en los módulos de Distribución de la información y Selectores de CPU y QPU. Durante el segundo mes, se desarrolló el Generador de Combinaciones. En el tercer mes, se implementó el servicio web para lanzar el Analizador de Coste-Rendimiento, así como la Calculadora de Utilidad. Finalmente, en el cuarto mes, se integraron los componentes desarrollados, se corrigieron errores y se finalizó la redacción de la memoria. Además, se han llevado a cabo reuniones quincenales con los tutores para revisar los avances.

En la Figura 4 se muestra un diagrama de la planificación del trabajo.

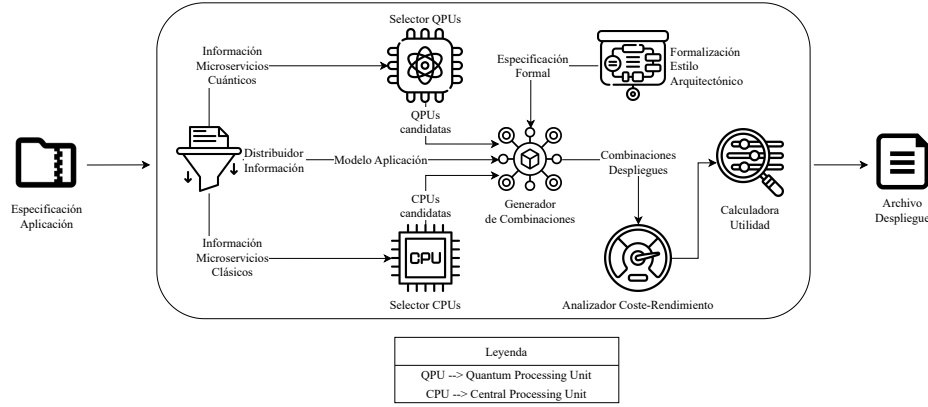
MES 1				MES 2			
Semana 1	Semana 2	Semana 3	Semana 4	Semana 5	Semana 6	Semana 7	Semana 8
T1				T1			
T2				T2			
	T3	T3			T3	T3	
		T4	T4			T4	T4
MES 3				MES 4			
Semana 9	Semana 10	Semana 11	Semana 12	Semana 13	Semana 14	Semana 15	Semana 16
T1				T1			
T2				T2			
	T3	T3			T3	T3	
		T4	T4			T4	T4

Fig. 4. Planificación del Trabajo

## 4.2 Diseño

La herramienta evaluadora que se presenta tiene como objetivo encontrar una distribución óptima en coste y rendimiento para aplicaciones híbridas (cuántico-clásicas). Para ello, se deben considerar dos aspectos clave. Primero, la selección de máquinas tanto clásicas como cuánticas más adecuada para el despliegue de los servicios que conforman la aplicación y segundo, la comprobación de las propiedades de rendimiento y coste asociado a las distintas configuraciones posibles para la distribución de los servicios cuánticos y clásicos.





**Fig. 5.** Sistema de Distribución de Aplicaciones Híbridas (Clásico-Cuánticas)

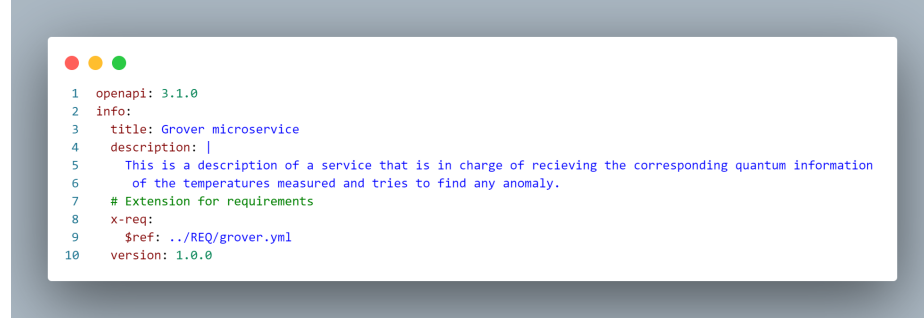
La Figura 5 representa la extensión de la ya presentada en el trabajo titulado “*Metodología para Desplegar Aplicaciones Híbridas (Cuántico-Clásicas)*” [42]. En ella se ilustra el proceso de trabajo diseñado para alcanzar la distribución óptima. Este proceso se inicializa con el análisis del archivo de entrada que detalla los servicios clásicos y cuánticos de la aplicación. Seguidamente, las especificaciones de cada tipo de servicio se asignan a selectores específicos, los cuales identifican un grupo de máquinas potenciales para alojar cada servicio. Posteriormente, considerando la definición del estilo arquitectónico, se crean diversas combinaciones de máquinas candidatas para evaluarlas en función de su coste y eficiencia. Después de analizar estas dos variables de la distribución, se calcula la utilidad de cada configuración, mostrando finalmente las distribuciones ordenadas por mayor valor de utilidad.

A continuación, se describe el diseño del comportamiento de cada módulo incluido en la metodología propuesta.

**Distribuidor de Información** Este módulo está encargado de recibir y procesar el archivo de especificación de la aplicación híbrida. Esta especificación contiene el modelo de la aplicación a evaluar, además de los requisitos hardware mínimos para la ejecución del servicio determinado, así como un conjunto de variables que sirven para definir el comportamiento que se encuentran recogidos en la Tabla 1. Esta definición de variables para describir los requisitos mínimos hardware y de comportamiento del servicio, están enfocadas a servir como extensión del estándar de definición de servicios OpenAPI<sup>6</sup>. El motivo de realizar la descripción del servicio y sus requisitos sea mediante una extensión de OpenAPI es porque éste, es considerado como un estándar “de facto” para el desarrollo de servicios [43]. Por tanto, es importante considerar su estructura para la nueva

<sup>6</sup> <https://www.openapis.org/>

generación de aplicaciones compuestas por servicios clásicos y servicios híbridos (clásico-cuánticos).



**Fig. 6.** Extensión de OpenAPI

Tras haber realizado la lectura de cada uno de los archivos de requisito de cada servicio, la información contenida es agrupada según el tipo, sea clásico o híbrido. Dicha información correspondiente a los servicios de tipo clásico se envía al selector de CPUs y la de tipo cuántico, se envía tanto al selector de CPU como el selector de QPU, ya que el servicio híbrido está compuesto por una parte clásica y otra parte cuántica. El modelo de la aplicación con sus restricciones y comportamiento es enviado al Generador de Combinaciones.

Cabe destacar que la variable de requisito *tiempo de ejecución* de un servicio híbrido será solamente la parte del servicio clásica para no diluir los tiempos de ejecución, puesto que debido al estado actual de acceso a los computadores cuánticos, al lanzar un algoritmo en algún proveedor de servicios, éste se almacena en una cola de ejecución cuyos tiempos de espera son muy alargados, motivado entre otros factores, porque los ordenadores cuánticos, no están todo el tiempo disponibles, sino que solo son accesibles durante su *ventana de accesibilidad*. Además, se espera que en un futuro próximo, según vayan proliferando las máquinas cuánticas, estos tiempos de cola se disminuyan tal y como pasó con los tiempos de cola al inicio de los supercomputadores.

**Selector de QPUs** Este componente, seleccionará el conjunto de computadoras cuánticas del proveedor de servicios, que por sus características físicas, se adecúan a la ejecución del algoritmo cuántico del servicio híbrido correspondiente. El filtro utilizado en esta metodología para descartar si una máquina cuántica puede o no ejecutar un determinado algoritmo es el siguiente:

Para cualquier máquina cuántica ( $m$ ) perteneciente al conjunto de máquinas cuánticas de un proveedor de servicios en la Nube ( $L$ ), la especificación es válida si se cumplen las siguientes condiciones:

1. El número de cubits del algoritmo cuántico  $a_q$ , debe estar dentro del rango de cubits soportados por la máquina cuántica en cuestión, es decir  $m_{q_{min}} \leq$

**Table 1.** Atributos de Especificación de Servicios

Nombre	Descripción	Exigidos
id	Cadena de caracteres que sirven para la identificación del servicio.	✓
api	URI para la especificación OpenAPI del servicio.	✓
modo	Indica el modelo de computación que utiliza el servicio (cuántico o clásico).	✓
memoria	Variable que describe la capacidad de memoria, en MB, requerida.	✓
cpu	El número de ciclos de CPU requeridos para la ejecución del servicio.	✓
tiempo_ejecución	Máximo tiempo de ejecución (segundos) de la tarea del servicio clásico.	×
cubits	Número de bits cuánticos utilizados en la implementación del algoritmo cuántico.	×
num_ejecuciones	Número de ejecuciones del algoritmo cuántico.	×
número_peticiones	Número de peticiones por minuto que recibe el servicio.	✓
tamaño_máximo_petición	El tamaño máximo en bytes de una petición.	✓
tiempo_ejecución	El tiempo máximo de ejecución de la tarea del servicio clásico.	✓
disponibilidad	Número de horas en las que el servicio está disponible al día.	✓
instancias	Número de instancias del servicio.	✓
obligatoriedad	Indica si el servicio debe aparecer en el despliegue	✓

\* Si el servicio es cuántico, los atributos *cubits* y *num\_ejecuciones* son obligatorios.

- $a_q \leq m_{q_{max}}$ , siendo  $m_{q_{min}}$  el número mínimo de cubits de la máquina  $(m)$  y  $m_{q_{max}}$ , el número máximo de cubits de la máquina  $(m)$ .
2. El número de *shots* (ejecuciones)  $a_s$  del algoritmo cuántico, debe estar dentro del rango de *shots* soportados por la máquina cuántica en cuestión, es decir  $m_{s_{min}} \leq a_s \leq m_{s_{max}}$  siendo  $m_{s_{min}}$  el número mínimo de *shots* de la máquina  $(m)$  y  $m_{s_{max}}$ , el número máximo de *shots* de la máquina  $(m)$ .

Formalmente podemos definirlo de la siguiente manera:

$$\forall m \in L, \quad ((m_{q_{min}} \leq a_q \leq m_{q_{max}}) \wedge (m_{s_{min}} \leq a_s \leq m_{s_{max}}))$$

El resultado de este selector es una colección de máquinas cuánticas compatibles por cada servicio híbrido. Siendo  $(hs_n)$  un servicio híbrido del conjunto de servicios híbridos  $(HS)$  de la aplicación híbrida  $(HA)$ , el resultado del selector de QPU es un listado de pares, en el que  $MQC_i$  es el conjunto de máquinas cuánticas compatibles con el servicio híbrido  $hs_i$ :

$$[(hs_1, MQC_1), (hs_2, MQC_2), \dots, (hs_n, MQC_n)], \text{ con } MQC_{i \in 1..n} \in 2^L$$

**Selector de CPUs** Este componente, seleccionará el conjunto de computadoras clásicas del proveedor de servicios, que por sus características hardware, se adecúan con las características descritas en la especificación del servicio. En este módulo, se seleccionarán también, las máquinas clásicas correspondientes al servicio híbrido. El filtro utilizado en esta metodología para descartar si una máquina cuántica puede o no ejecutar un determinado algoritmo es el siguiente: Para cualquier máquina clásica  $(c)$  perteneciente al conjunto de máquinas clásicas de un proveedor de servicios  $(J)$ , la especificación es válida si se cumplen las siguientes condiciones:

1. La mínima memoria RAM para el servicio, debe ser inferior o igual al valor de memoria RAM de la máquina en cuestión, es decir,  $s_r \leq c_r$  siendo  $s_r$  el valor mínimo de RAM del servicio y  $c_r$  el valor de RAM de la máquina  $c$ .
2. El ancho de banda mínimo para el servicio, debe ser inferior al valor de ancho de banda de la máquina en cuestión, es decir,  $s_b \leq c_b$  siendo  $s_b$  el valor mínimo de ancho de banda del servicio y  $c_b$  el valor de ancho de banda de la máquina  $c$ .
3. La capacidad de procesamiento mínima para el servicio, debe ser inferior al valor del capacidad de procesamiento de la máquina en cuestión, es decir,  $s_c \leq c_c$  siendo  $s_c$  el valor mínimo de capacidad de procesamiento del servicio y  $s_c$  el valor de capacidad de procesamiento de la máquina  $c$ .

El resultado de este selector de CPU, es una colección de máquinas clásicas compatibles por cada servicio de la aplicación. Siendo  $(s_n)$  un servicio del conjunto de servicios  $(S)$  de la aplicación híbrida  $(HA)$ , el resultado del selector de CPU es un listado de pares en el que  $MCC_i$  es el conjunto de máquinas clásicas compatibles con el servicio  $c_i$ :

$$[(s_1, MCC_1), (s_2, MCC_2), \dots, (s_n, MCC_n)], \text{ con } MCC_{i \in 1..n} \in 2^J$$

**Generador de Combinaciones** Este módulo, recibe como entrada la selección de máquinas clásicas capaces de alojar los servicios clásicos de la aplicación así como la selección de máquinas cuánticas adecuadas para la ejecución del algoritmo cuántico del servicio híbrido, procedentes de los selectores de CPU y QPU respectivamente, además del modelo con las restricciones y comportamiento de la aplicación híbrida.

A su vez, este generador hace uso de la especificación formal del estilo arquitectónico de las aplicaciones híbridas (clásico-cuánticas) definida por expertos en el dominio. Esta especificación, describe de forma abstracta el conjunto de firmas y restricciones necesarias para generar las diferentes posibilidades de despliegue de los servicios que conforman cualquier aplicación híbrida. Estos diferentes despliegues se crean mediante el uso de un *Constraint Solver* (solucionador de restricciones, por ejemplo Alloy <sup>7</sup>). Sin embargo, para poder estudiar diferentes propiedades cuantitativas de los modelos, es necesario complementar al *constraint solver* con una herramienta que mediante la especificación del comportamiento complete las estructuras arquitectónicas de las alternativas de despliegue, ofreciendo como resultado un conjunto de modelos de configuraciones de despliegue con comportamientos evaluables a través del *model checker probabilístico*. Resulta probable que se produzca una explosión combinatoria debido al número de posibles máquinas y servicios. Es por ello por lo que será preciso aplicar técnicas para el cribado de máquinas y así el proceso de evaluación sea abordable.

**Analizador de Coste y Rendimiento** Este analizador emplea una herramienta de verificación probabilística, como PRISM <sup>4</sup>, para evaluar las dimensiones de coste y rendimiento de las diversas configuraciones de despliegue de los servicios de la aplicación que recibe como entrada.

Para calcular estas métricas, es esencial considerar diversos aspectos interrelacionados. No basta con evaluar únicamente el coste económico de los componentes utilizados; es igualmente necesario tener en cuenta cómo estos componentes se interconectan y su nivel de fiabilidad. Por ejemplo, al utilizar un servicio híbrido para un cálculo específico, podría suceder que este no sea completamente fiable y presente problemas con frecuencia. En tal caso, tanto el coste como el rendimiento deberían verse afectados en comparación con un servicio de características similares pero con mayor fiabilidad.

Por ello, el uso combinado de verificación probabilística y un *constraint solver* permite resolver el problema del cálculo de coste y rendimiento al considerar todos los factores involucrados en estas métricas.

El resultado de este proceso es una colección de valores de rendimiento y coste del sistema para cada posible distribución de despliegue.

En el caso de los servicios híbridos, el cálculo del rendimiento en las máquinas cuánticas no puede realizarse de la misma manera que en un ordenador clásico, debido a que la ejecución de un algoritmo cuántico debe esperar en una cola

<sup>7</sup> <https://alloytools.org/>

<sup>4</sup> <https://www.prismmodelchecker.org/>

de procesos que avanza conforme el ordenador cuántico se encuentra disponible. Por esta razón, la dimensión de rendimiento no puede ser fija. En esta situación, la verificación probabilística permite establecer una variable aleatoria que siga una distribución similar al tiempo que tarda en estar disponible una máquina cuántica en los proveedores de servicios en la nube y al tiempo que toma la ejecución del algoritmo cuántico. No obstante, para la implementación de este trabajo, se ha optado por considerar un valor neutro de rendimiento para los servicios cuánticos.

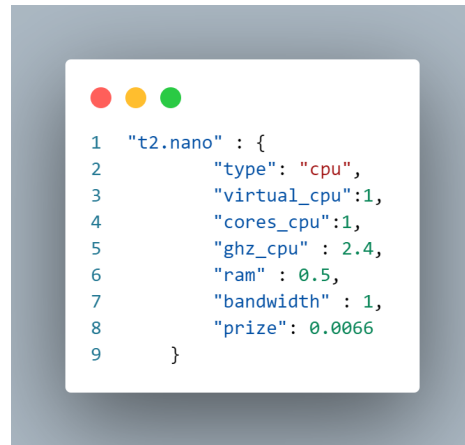
**Calculadora de Utilidad** Este componente es responsable de calcular el valor de utilidad para cada una de las soluciones eficientes de Pareto, es decir, el conjunto de distribuciones de la aplicación con sus respectivos valores de rendimiento y coste económico. En este módulo se utiliza una función de utilidad basada en la suma ponderada, donde se otorga una mayor importancia, previamente definida por el usuario, a una de las dimensiones (coste o rendimiento). Esta calculadora evalúa cada despliegue, determinando su valor de utilidad, y produce como resultado un archivo que contiene la combinación de distribución que maximiza dicha función de utilidad.

### 4.3 Implementación

En esta sección se detallarán los aspectos de la implementación necesarios para el desarrollo del Sistema Evaluador de Coste para Aplicaciones Híbridas (Cuánticas-Clásicas) así como las tecnologías utilizadas. Tomando como referencia la arquitectura presentada en la Figura 5, a continuación, se irá exponiendo cada elemento, desde la creación de los archivos de especificación, hasta la salida del sistema, con la obtención del archivo de despliegue. En esta primera versión de la herramienta evaluadora, se ha utilizado la información de las máquinas, tanto clásicas como cuánticas, que ofrece el proveedor de servicios cloud, Amazon Web Services (AWS) <sup>8</sup>. Sin embargo, AWS ofrece más de **400** modelos de máquinas clásicas por lo que para que no se produzca una explosión combinatoria, se ha procedido a limitar tanto el número de máquinas como la información que ofrece con el fin de tener un conjunto limitado y con las características que inicialmente esta herramienta necesita para lograr su objetivo. Entre las limitaciones destaca, además del número total de máquinas clásicas que pasan a ser **14**, la selección solamente de las versiones Linux. También, se ha llevado a cabo una transformación del ancho de banda de aquellas máquinas que indicaban como ancho de banda *Low to Moderate* a 1Mbps tomando como referencia los resultados que se indican en el trabajo de Mikyung Kang et al. [44]. Para las máquinas cuánticas, se han tomado únicamente la información relacionada con los aspectos que se evalúan (precio, rango de shots y número de cubits).

En consecuencia, se han generado dos archivos de tipo *JSON*, uno que contiene la información de las máquinas clásicas y otro para las máquinas cuánticas. Pueden verse una muestra de ambos archivos en las Figuras 7 y 8.

<sup>8</sup> <https://aws.amazon.com/es/>



**Fig. 7.** Extracto de JSON de máquinas clásicas.

A continuación, se explican las tecnologías utilizadas así como el desarrollo de los módulos que componen el sistema de evaluación.

**Tecnologías utilizadas** Las tecnologías utilizadas para la implementación de este sistema han sido las siguientes:

- **Python:** Lenguaje de programación que permite el trabajo rápido y la integración de sistemas de forma efectiva <sup>9</sup>. Es ideal para el desarrollo de prototipos y pruebas de concepto, por lo que en este trabajo se ha utilizado para el desarrollo funcional de cada uno de los módulos que componen el sistema de evaluación, a excepción del analizador de coste y rendimiento. Concretamente, se ha utilizado para el desarrollo la versión de Python 3.11.9.
- **Javascript:** Lenguaje de programación orientado a objetos y especialmente utilizado en el desarrollo web para la dinámica de contenidos <sup>10</sup>. En este trabajo se ha utilizado como elemento para el desarrollo de la interfaz web.
- **HTML5:** Lenguaje de Marcado de Hipertexto (en inglés, HyperText Markup Language) usado para la definición de la parte estática de una web <sup>11</sup>. Ha sido empleado en la implementación del sistema para la interfaz web.
- **Jinja2:** Lenguaje de plantillas para Python <sup>12</sup>. Se ha utilizado para el desarrollo de la interfaz web.
- **Bootstrap:** Framework estilográfico para el desarrollo de interfaces de usuario en el lado del cliente <sup>13</sup>. Usado en la implementación de la interfaz web.

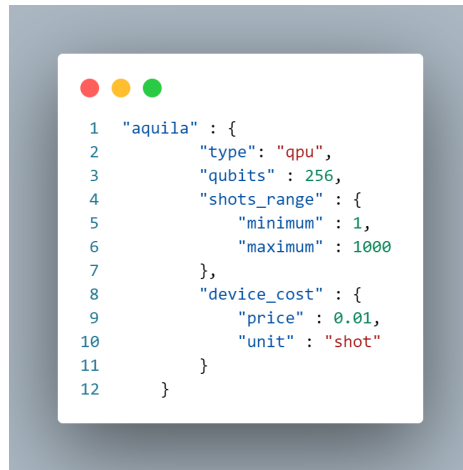
<sup>9</sup> <https://www.python.org/>

<sup>10</sup> <https://developer.mozilla.org/es/docs/Learn/JavaScript>

<sup>11</sup> <https://developer.mozilla.org/es/docs/Web/HTML>

<sup>12</sup> <https://jinja.palletsprojects.com/en/2.10.x/>

<sup>13</sup> <https://getbootstrap.com/>



```
1  "aquila" : {
2      "type": "qpu",
3      "qubits" : 256,
4      "shots_range" : {
5          "minimum" : 1,
6          "maximum" : 1000
7      },
8      "device_cost" : {
9          "price" : 0.01,
10         "unit" : "shot"
11     }
12 }
```

**Fig. 8.** Extracto de JSON de máquinas cuánticas.

- **Flask:** Microframework para el desarrollo de prototipos de servicios web, y APIs en Python <sup>14</sup>. Se ha utilizado para el desarrollo de los módulos del sistema evaluador.
- **Apache Kafka:** Plataforma para la distribución de eventos en tiempo real, mediante el protocolo de comunicación Pub-Sub <sup>15</sup>. Esta herramienta ha sido utilizada para establecer la comunicación y envío de mensajes entre distintos módulos del sistema.
- **Docker:** Tecnología para la contenedorización de aplicaciones y respectivas dependencias <sup>16</sup>. En particular, se ha utilizado una extensión de Docker que permite el manejo de aplicaciones multicontenedor, llamada Docker Compose, para la gestión de Apache Kafka.
- **VirtualBox:** Software de virtualización para la ejecución de múltiples sistemas operativos en máquinas virtuales <sup>17</sup>. Se ha utilizado para la creación de dos máquinas virtuales para el despliegue de los componentes del sistema.
- **Alloy:** Lenguaje y sistema para el análisis de modelos software <sup>18</sup>. Ha sido utilizado para definir el modelo y restricciones de las aplicaciones de servicios híbridas así como para la especificación de la aplicación utilizada en la fase de evaluación.
- **HaiQ:** Herramienta que mejora el modelado/síntesis estructural con garantías cuantitativas al estilo que proporciona la verificación cuantita-

<sup>14</sup> <https://flask.palletsprojects.com/en/3.0.x/>

<sup>15</sup> <https://kafka.apache.org/>

<sup>16</sup> <https://www.docker.com/>

<sup>17</sup> <https://virtualbox.org>

<sup>18</sup> <https://alloytools.org/>



tiva <sup>19</sup>. Esta herramienta se ha utilizado para llevar a cabo el análisis del coste/rendimiento para cada posibilidad de despliegue.

**Servicio Web** Debido a que nuestro sistema necesita de un conjunto de archivos de especificación desarrollados por el usuario así como el establecimiento de los valores de los pesos de las variables coste y rendimiento según la preferencia del mismo, se ha optado por desarrollar un servicio web con interfaz para que resulte más sencillo al usuario la utilización del sistema.

Esta interfaz web, tal y como se puede observar en la Figura 9, cuenta con un formulario donde el usuario debe adjuntar, tanto las especificaciones OpenAPI como el modelo de la aplicación con las respectivas restricciones y comportamientos para servicio. Todo ello comprimido en un archivo *.zip* siguiendo la estructura que se indica en la siguiente sección. Además, a través de esta web, el usuario puede indicar los pesos de las variables a estudiar, en este caso, coste y rendimiento, teniendo en cuenta que estos valores deben estar normalizados entre 0 y 1 y que la suma de estos pesos debe de ser igual a uno.

**Sistema de Evaluación de Coste para Aplicaciones Híbridas (Cuánticas-Clásicas)**

Los valores de los pesos de coste y rendimiento deben de estar en el rango de 0.0 y 1.0.  
Siendo la suma de ambos pesos = 1.

**Cost Weight**  
Cost Weight

**Performance Weight**  
Performance

**App Specification**  
Browse... No file selected.

**Enviar**

© 2024 Trabajo Fin de Master - Alvaro M. Aparicio-Morales.  
Tutorizado por: Jose Garcia-Alonso y Javier Camara  
Master Universitario en Ingeniería Informática (mención Ingeniería y Ciencia de Datos)  
Universidad de Málaga, Septiembre 2024  
Todos los derechos reservados.

**Fig. 9.** Página principal de la Interfaz.

Con el fin de comenzar el proceso de evaluación, tras adjuntar el archivo *.zip* y los valores de los pesos, se debe de presionar el botón *Enviar* que se encargará de enviar el formulario al servicio web para que envíe el archivo y los pesos a la ruta del distribuidor de información */start*.

**Distribuidor de Información** Este servicio se encarga de recibir la información enviada a través del formulario del servicio web así como de recibir la solución del Analizador de Coste y Rendimiento.

<sup>19</sup> <https://github.com/javier-camara/HaiQ>

El flujo de éste módulo, comienza con la lectura de los valores de utilidad, que son almacenados temporalmente hasta que se recibe la solución del Analizador. Tras ello, comienza con el procesamiento del archivo comprimido que debe de estar formado por tres carpetas tal y como se puede observar en la Figura 10. Cada una de estas carpetas contienen la siguiente información:

 ARC	Carpeta de archivos
 OAS	Carpeta de archivos
 REQ	Carpeta de archivos

**Fig. 10.** Estructura carpetas .zip

- **ARC:** Esta carpeta contiene dos archivos. El primero de ellos es la especificación del comportamiento de la aplicación y el segundo es la especificación de las restricciones de la aplicación.
- **OAS:** En su interior se encuentra la especificación OpenAPI de cada servicio que compone la aplicación.
- **REQ:** Alberga para cada servicio, la extensión del estándar de OpenAPI propuesto para este trabajo.

Con el uso de la especificación OpenAPI, se permite definir la lógica de negocio que realiza el servicio definido permitiendo además generar extensiones de este estándar. Sin embargo, dada la dimensión del problema de “ubicación de servicios” de aplicaciones híbridas, que pretende abordar este trabajo, es necesario incluir en esta especificación, los requisitos hardware necesarios para lograr seleccionar aquellas máquinas compatibles con los servicios. Este contenido es el que se encuentra dentro de la carpeta *REQ*.

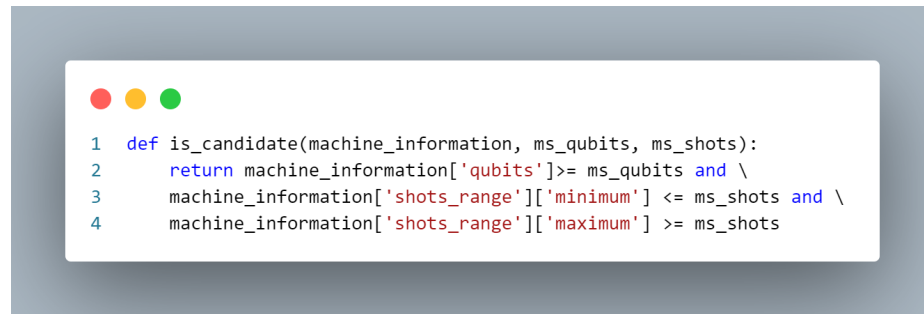
Para poder relacionar la extensión propuesta con la especificación de OpenAPI, utilizamos el atributo de definición de OpenAPI *info* para definir un nuevo atributo denominado *x-req* que contiene la ruta del archivo de requisitos del servicio. En la Figura 6, puede observarse dicho atributo dentro de un ejemplo de especificación del estándar OpenAPI.

En el procesamiento de los archivos de requisitos de los servicios, se construyen dos diccionarios. Uno con la parte cuántica de los servicios híbridos y el segundo diccionario, contiene los servicios clásicos así como la parte clásica de los servicios híbridos. Sendos diccionarios, son enviados a sus correspondientes selectores. La estructura de ambos diccionarios es similar, contiene un diccionario por cada servicio de forma que las claves son el id del servicio y los valores son el resto de variables que se definen en la Tabla 1.

Además, en este módulo, también se produce la lectura del contenido de la carpeta ARC, que es enviado directamente al componente Generador de Combinaciones, y la lectura del resultado del Analizador a través de la ruta */haiq-result*.

Tras recibir el resultado, éste se envía junto con los valores de los pesos del coste y rendimiento a la Calculadora de Utilidad.

**Selector de QPUs** Este módulo recibe el diccionario de pares servicio-requisitos de los servicios cuánticos que forman parte de la aplicación híbrida y realiza un procesamiento por cada servicio aplicando el filtro de selección expuesto formalmente en la fase de diseño a través de la llamada a la función *is\_candidate* que puede verse en la Figura 11.



```

1 def is_candidate(machine_information, ms_qubits, ms_shots):
2     return machine_information['qubits'] >= ms_qubits and \
3         machine_information['shots_range']['minimum'] <= ms_shots and \
4         machine_information['shots_range']['maximum'] >= ms_shots

```

**Fig. 11.** Función de filtrado de QPUs.


El resultado es una salida en formato json que contiene el nombre del servicio y el conjunto de máquinas candidatas.

**Selector de CPUs** Este módulo recibe el diccionario de pares servicio-requisitos de los servicios clásicos así como de la parte clásica que forman parte de la aplicación híbrida y realiza un procesamiento por cada servicio aplicando el filtro de selección expuesto formalmente en la fase de diseño a través de la llamada a la función *is\_candidate* que puede verse en la Figura 12.

En esta figura, puede observarse que se realiza una comparativa con una variable denominada *ms\_logical\_performance\_cpu*. Esta variable, representa un concepto de métrica que ha tenido que ser creada para obtener una estimación, más o menos, realista de la potencia computacional requerida por el servicio, debido a la existencia núcleos físicos y virtuales en las características de las máquinas. En este caso, la variable *ms\_logical\_performance\_cpu*, representa el factor lógico de rendimiento del servicio que se calcula tal y como se expresa en la Figura 13.

Para el caso de la especificación del servicio, las **CPUs Virtuales** y **Núcleos** tendrán el valor de 1.

En el caso de las máquinas, la métrica de potencia computacional de la CPU, se realiza mediante una multiplicación de *número\_de\_cpus\_virtuales*  $\times$  *núcleos*  $\times$  *frecuencia*. Esta multiplicación puede verse también en la Figura 12 y será




```

1 def is_candidate(machine_information, ms_logical_performance_cpu, ms_ram, ms_bandwidth):
2     machine_logical_performance_factor = (float(machine_information['virtual_cpu'])*\
3                                           float(machine_information['cores_cpu'])*\
4                                           float(machine_information['ghz_cpu']))
5     return machine_logical_performance_factor >= ms_logical_performance_cpu \
6           and float(machine_information['ram']) >= float(ms_ram) \
7           and int(machine_information['bandwidth']) >= ms_bandwidth

```

Fig. 12. Función de filtrado de CPUs.



```

1 # From Cycles per second to GHz * 1 Core * 1 Virtual CPUs
2 mslogical_performance_ms_factor = requirements['cpu'] / 1_000_000_000

```

Fig. 13. Transformación a factor lógico de rendimiento.

la métrica establecida como potencia computacional de una máquina candidata que, para cumplir la restricción, deberá ser mayor que la potencia computacional requerida por el servicio. El razonamiento por el cual se decidió crear esta métrica para estimar la potencia computacional requerida por el servicio y la de cada máquina es el siguiente:

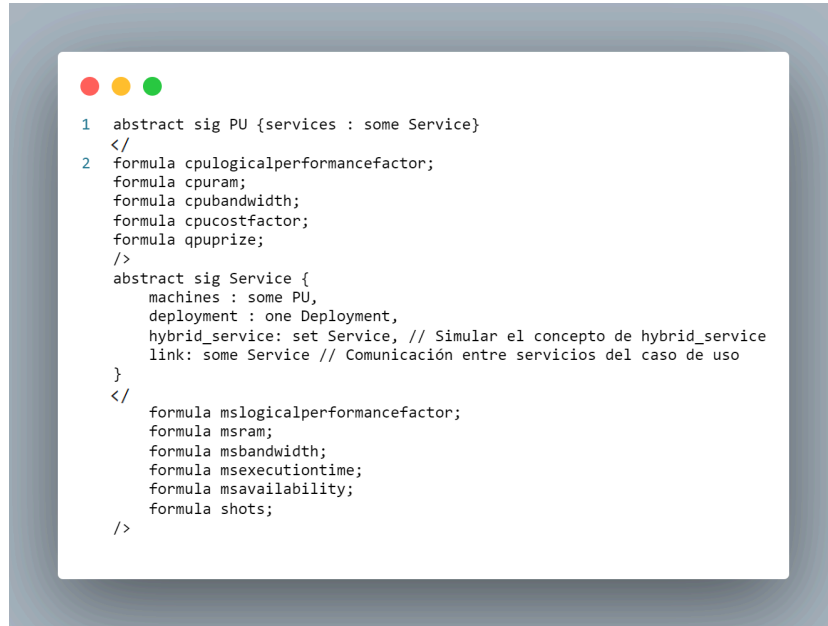
*“Al poseer cada máquina uno o más núcleos, significa que cada núcleo va a procesar la información a la frecuencia que determina el atributo de frecuencia de las características de la máquina. Por consiguiente, la “frecuencia total”, es decir, la suma de las frecuencias por núcleo, sería núcleos\*frecuencia. Además, el término de cpus virtuales, indica el número de núcleos lógicos que puedo utilizar por cada núcleo físico, por tanto, la métrica de potencia lógica computacional de mi máquina será la multiplicación de los tres factores, cpus\_virtuales × núcleos × frecuencia.”*

El resultado es una salida en formato json que contiene el nombre del servicio y el conjunto de máquinas candidatas.

**Generador de Combinaciones** Para generar las diferentes combinaciones de despliegues se necesita un archivo que especifique el estilo arquitectónico de las configuraciones de despliegues de las aplicaciones híbridas (clásico-cuánticas) que se ha desarrollado para la elaboración de este trabajo. En el Apéndice A

de esta memoria, se encuentra la descripción completa del estilo arquitectónico para aplicaciones híbridas.

En el interior de este archivo, se encuentra el modelo junto con las restricciones generales de este tipo de aplicaciones. En las Figuras 14 y 15 pueden observarse dos fragmentos de este archivo.



```

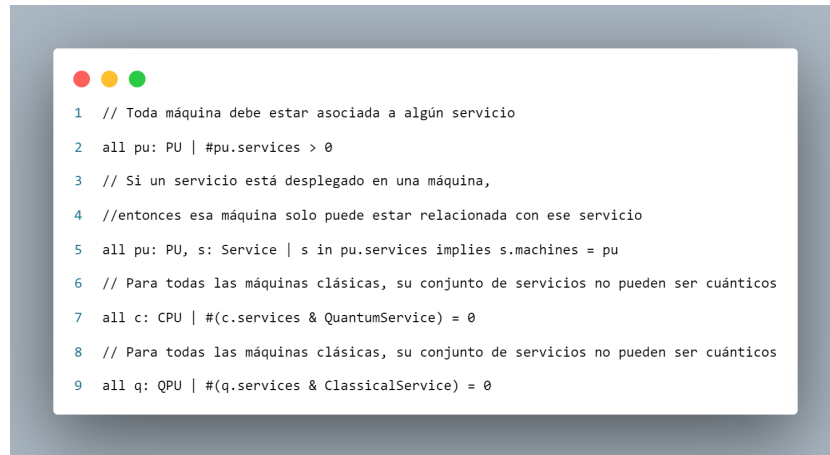
1  abstract sig PU {services : some Service}
  </
2  formula cpulogicalperformancefactor;
  formula cpuram;
  formula cpubandwidth;
  formula cpucostfactor;
  formula qpuprize;
  />
  abstract sig Service {
    machines : some PU,
    deployment : one Deployment,
    hybrid_service: set Service, // Simular el concepto de hybrid_service
    link: some Service // Comunicación entre servicios del caso de uso
  }
  </
  formula mslogicalperformancefactor;
  formula msram;
  formula msbandwidth;
  formula msexecutiontime;
  formula msavailability;
  formula shots;
  />

```

**Fig. 14.** Extracto de Estilo Arquitectónico (Signaturas).

Por tanto, con este archivo de definición del estilo arquitectónico de las aplicaciones híbridas, junto con el modelo de la aplicación proporcionado por el usuario, así como con las salidas de los selectores, este componente generador crea de forma automatizada, el modelo junto con las restricciones y las propiedades que se quieren estudiar, en este caso son el coste y rendimiento (Figura 16). Este modelo resultante, es enviado al Analizador de Coste y Rendimiento a través de una petición http a la ruta */launch-haiq*.

**Analizador de Coste y Rendimiento** Este módulo se encarga de recibir el modelo generado por el Generador de Combinaciones, almacenarlo y realizar el lanzamiento de la herramienta HaiQ. Tras la ejecución de HaiQ, se obtiene un archivo con todos los posibles despliegues generados en la herramienta en base a las restricciones definidas en el modelo generado junto con los valores de coste y rendimiento de cada despliegue. Este archivo de solución, es enviado al módulo



**Fig. 15.** Extracto de Estilo Arquitectónico (Restricciones).

de procesamiento de información que se encarga de recibir y almacenar de forma temporal dicho archivo.

**Calculadora de Utilidad** Este módulo recibe desde el componente *Distribuidor de Información*, a través de un mensaje de Kafka, el contenido del archivo de soluciones generado por HaiQ. Esta calculadora de utilidad, realiza un procesamiento del contenido mediante un proceso iterativo analizando cada uno de los despliegues generados y calculando el valor de utilidad de cada uno de ellos. A la vez que se calcula su valor de utilidad, se procede a añadir dentro de una lista de tuplas llamada *utility\_tuple\_sorted\_list* el valor de utilidad de dicha solución, así como sus valores de coste y rendimiento.

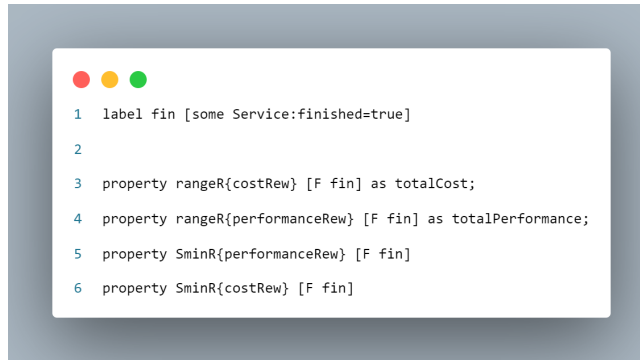
Tras obtener la lista completa de tuplas, ésta se envía al *Servicio Web* para mostrar dichos resultados ordenados por valor de utilidad, como puede apreciarse en la Figura 20.

#### 4.4 Evaluación

Con el fin de comprobar el sistema de evaluación de coste y rendimiento de aplicaciones híbridas (clásico-cuánticas) se ha propuesto el siguiente caso de estudio. Este caso de estudio consiste en una aplicación de IoT en la cuál se verifica si ha existido una anomalía en el valor de los sensores. Esta aplicación está compuesta de los siguientes servicios:

- **Servicio de Datos:** Este servicio, se encarga de recibir los datos del sensor de temperatura que le corresponde y los envía al servicio *Agregador*. Para este caso de uso se han definido 3 instancias de este servicio de datos.
- **Servicio Agregador:** Este servicio se encarga de recibir, unificar y transformar los datos de temperatura a normalizados de 0 ó 1. Tras esto, los datos son enviados al servicio de Búsqueda.

- **Servicio de Búsqueda:** Este servicio se encargará de procesar los datos recibidos para detectar si ha existido una anomalía o no. Para este servicio de búsqueda existen dos alternativas. Por un lado, tendremos un servicio de búsqueda clásico que implementa un algoritmo de búsqueda binaria. Por otro lado, tendremos un servicio de búsqueda híbrido que implementa el algoritmo de Grover, siendo el *equivalente* cuántico del algoritmo de búsqueda binaria. El algoritmo de *Grover*, es un algoritmo cuántico que se encarga de determinar si un elemento se encuentra en una lista no ordenada de elementos de tamaño  $N$  [4]. A diferencia de la búsqueda binaria, la complejidad de *Grover* es  $O(\sqrt{N})$ . Finalmente, tras ejecutar la tarea de búsqueda, ambos servicios enviarían sus resultados al *Servicio de Procesamiento*.
- **Servicios de Procesamiento:** Este servicio de procesamiento se encarga de recibir y analizar la respuesta otorgada por el servicio de búsqueda. En caso de haber detectado alguna anomalía, lo notifica al usuario.



**Fig. 16.** Propiedades Cuantitativas a Estudiar

Como se ha expuesto anteriormente, el servicio de búsqueda puede ser o bien clásico o bien híbrido. Dado el contexto en el que se halla la computación cuántica el flujo natural de la aplicación varía significativamente de un servicio de búsqueda u otro. A continuación, en las Figuras 18 y 17 se pueden observar el flujo de cada servicio.

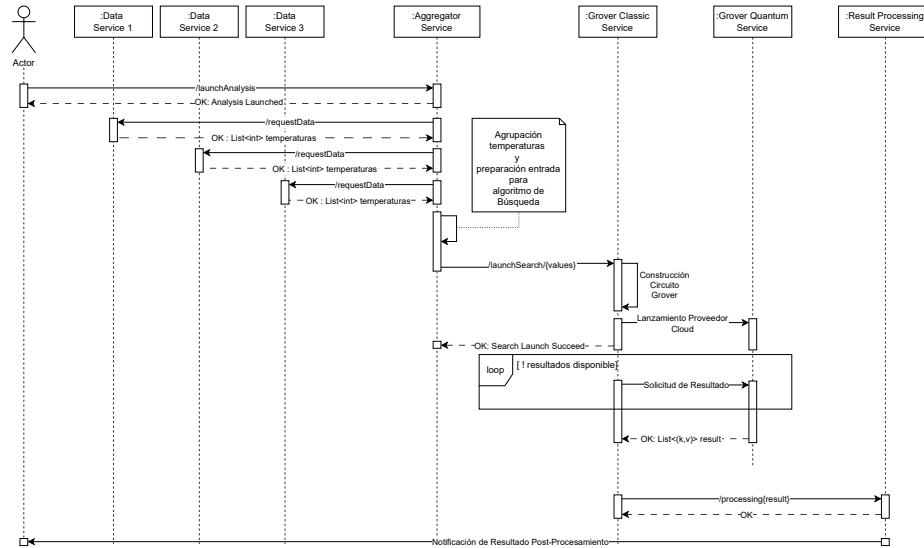
El objetivo de esta evaluación es comprobar qué distribuciones, según los parámetros del usuario, son las más adecuadas. A fin de poder realizar la evaluación del sistema, es necesario implementar la especificación formal y de comportamiento así como la especificación OpenAPI y de los requisitos de los distintos servicios que componen la aplicación IoT.

En el desarrollo de la especificación formal y de comportamiento de la aplicación, se ha utilizado el lenguaje de modelado de Alloy junto con la sintaxis específica de HaiQ para la definición del comportamiento.

Tras haber definido ambas especificaciones, éstas deben de comprimirse en un archivo ZIP, que contenga la estructura presentada en la Figura 10.

Una vez generado el archivo *.zip*, en la interfaz web introducimos las preferencias de pesos para cada una de las variables a estudiar, en este caso coste y rendimiento, y adjuntamos el archivo *.zip* (Figura 9). Tras enviar la información, el sistema irá ejecutando módulo a módulo hasta que tras finalizar el análisis de coste-rendimiento, podremos observar y descargar los resultados tal y como puede apreciarse en la Figura 20.

Finalmente, se puede descargar el archivo de salida, el cuál, contendrá un diccionario donde a cada servicio le corresponde una determinada máquina como puede observarse en las Figuras 21 y 22 que corresponden con la parte clásica y parte cuántica del algoritmo de Grover respectivamente.



**Fig. 17.** Diagrama de Flujo de Búsqueda Híbrida



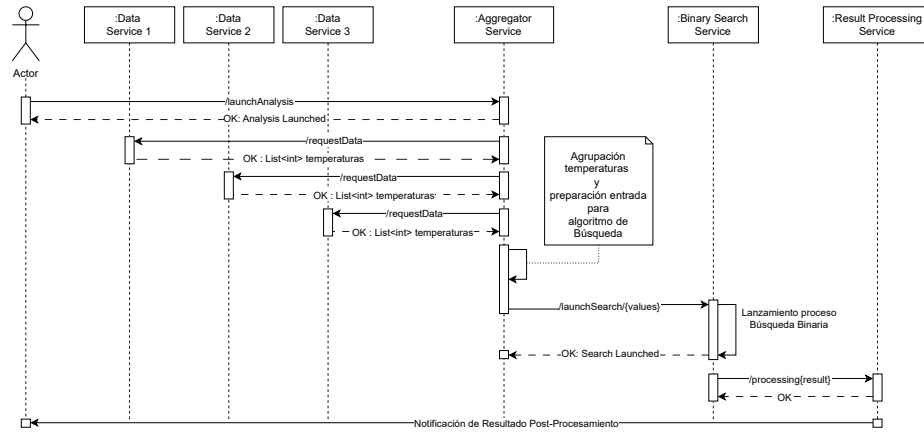


Fig. 18. Diagrama de Flujo Búsqueda Clásica



Fig. 19. Fases del Sistema de Evaluación

Sistema de Evaluación de Coste para Aplicaciones Híbridas

(Cuánticas-Clásicas)

Cost Weight

0.5

Performance Weight

0.5

Home

Show 10 entries

Search:

Solution Number	Utility Value	Cost	Performance	
sol_6	3964299.645098412	3.9648000000000003	7928595.325396824	Download Distribution
sol_18	3496580.143158024	4.1536	6993156.1327160485	Download Distribution
sol_7	2982159.647612698	9.82697142857143	5964309.468253967	Download Distribution
sol_5	2714298.904869841	2.912914285714286	5428594.896825396	Download Distribution
sol_4	2642878.5832507936	19.793485714285715	5285737.373015873	Download Distribution
sol_20	2642878.24759365	19.550742857142858	5285736.944444443	Download Distribution
sol_16	2625013.399398412	3.4734	5250023.325396825	Download Distribution
sol_10	2535727.3214412695	3.3174857142857146	5071451.325396825	Download Distribution
sol_12	2527829.0625358024	4.4368	5055653.6882716045	Download Distribution
sol_13	2375012.500107936	2.9129142857142862	4750022.087301587	Download Distribution

Showing 1 to 10 of 25 entries

Previous

1

2

3

Next

Fig. 20. Resultados Sistema Evaluador

```

1  {
2      "groveralg00": [
3          {
4              "deployment": "HybridUseCase0"
5          },
6          {
7              "hybrid_service": "groveralg0"
8          },
9          {
10             "link": "groveralg0"
11          },
12          {
13             "machines": "t3large0"
14          },
15          {
16             "link": "resultprocessing00"
17          },
18          {
19             "link": "aggregator00"
20          }
21      ]
22  }
```

Fig. 21. Extracto Solución (Parte Clásica del Servicio Híbrido)



```

1  {
2      "groveralg0": [
3          {
4              "machines": "aria10"
5          },
6          {
7              "link": "groveralg00"
8          },
9          {
10             "hybrid_service": "groveralg00"
11          },
12          {
13             "deployment": "HybridUseCase0"
14          }
15      ]
16  }

```

**Fig. 22.** Extracto Solución (Parte Cuántica del Servicio Híbrido)

#### 4.5 Trabajos relacionados

Esta sección, se organiza en torno a dos áreas que constituyen el núcleo de este trabajo. Por un lado, la computación cuántica, mediante la creación de aplicaciones híbridas de servicios, donde coexisten servicios clásicos y cuánticos. Por otro lado, la resolución del *placement problem* para aplicaciones de servicios clásicas dentro del ámbito de la arquitectura de servicios. En primer lugar, se abordarán aquellos estudios de aplicaciones de servicios híbridas cuántico-clásicas. En segundo lugar se tratarán los trabajos existente que resuelven la obtención de un despliegue óptimo de aplicaciones de servicios clásicas.

**Computación Cuántica** En el ámbito de la computación cuántica, podemos resaltar aquellos trabajos que tengan como objetivo, la elección de un computador para un algoritmo cuántico como es el caso del estudio de Sivarajah et al. [45] (t|ket) y JavadiAbhari et al. [46] (ScaffCC), que validan si una implementación algorítmica es ejecutable en una computadora cuántica específica o el trabajo titulado *NISQ Analyzer* [1], que sugiere la implementación óptima de un algoritmo y la computadora cuántica adecuada para ciertos datos. No obstante, la decisión final sigue siendo responsabilidad del usuario.

En estudios centrados en la coexistencia de servicios clásicos y cuánticos se destacan el trabajo de Romero et al. [47] en el que se realiza una propuesta de planificador para la ejecución de circuitos cuánticos en los proveedores de servicios en la nube, o el elaborado por Alvarado et al. [48] donde se presenta una guía para la conversión de circuitos cuánticos a servicios web así como el estudio de Nguyen et al. [49] donde se presenta el concepto de *quantum function-as-a-service*, donde se usa el modelo *serverless* para la ejecución de circuitos cuánticos.

**Arquitectura de Servicios** Estos estudios se centran en la resolución del *placement problem* así como en el estudio de propiedades en el comportamiento de las aplicaciones, cuestiones clave para optimizar el despliegue de aplicaciones de servicios, maximizando o minimizando algunas propiedades cuantitativas del sistema distribuido.

Actualmente, existen diversas herramientas y plataformas que ofrecen recomendaciones basadas en la composición de aplicaciones de servicios y los requisitos del usuario como es el caso del trabajo de Herrera et al. [15] o el estudio de Yang Hu et al. [50] donde se busca optimizar el despliegue de aplicaciones basadas en microservicios en entornos de nube minimizando el tráfico entre componentes o el estudio de Selimi et al. [51] cuyo objetivo es aprovechar la información de estado sobre la red para fundamentar las decisiones de colocación de servicios. Estas son algunas de las propiedades que pueden ser evaluadas de forma similar a lo que se propone en el trabajo de Cámara et al. [52] en el cual se analiza el comportamiento de aplicaciones bajo incertidumbre utilizando verificación formal probabilística.

Estos son algunos ejemplos de trabajos que, de forma afín, están relacionados con lo que se presenta y que han servido como base para el objetivo de este trabajo en el que se unifica de ambos ámbitos proponiendo una novedosa herramienta en este aspecto, ya que hasta donde sabemos, no existen trabajos que aborden el *placement problem* para aplicaciones híbridas cuántico-clásicas desde la perspectiva propuesta en este estudio.

## 5 Conclusiones

El nuevo paradigma de computación cuántico, está llamado a regentar una revolución en la computación gracias a su característica de alta capacidad computacional que le proporciona las propiedades cuánticas de superposición y entrelazamiento cuántico. Esta característica abre camino para lograr su aplicación en el afrontamiento de nuevos retos en diferentes ámbitos científicos y tecnológico, como puedan ser: ámbito biomédico, mediante el desarrollo de nuevos fármacos y medicina personalizada [6]; ámbito industrial, mejorando el mantenimiento autónomo impulsado por gemelos digitales [53]; o en la resolución de problemas de optimización y redistribución de recursos; e incluso en el área de la química, con el descubrimiento de nuevos materiales [54], son algunos de los distintos usos que se espera que estos nuevos computadores tengan. Estos son solo algunos de los usos que se espera que tengan estos nuevos ordenadores. Sin embargo, para llegar a su aplicación, todavía es necesario un largo recorrido en el desarrollo tanto de hardware como de soluciones que sean capaces de aprovechar este nuevo modelo computacional. Este camino no está exento de desafíos, entre los que destaca, la manera de conseguir la integración de estos computadores en los sistemas de información actuales. Este reto, puede ser abordado desde el área de la Ingeniería de Software mediante la Ingeniería de Servicios.

La aplicación de la Ingeniería de Servicios para la integración de la computación cuántica, propone que ésta sea a través de la interacción de los servicios

clásicos y de los servicios cuánticos. Sin embargo, la computación orientada a servicios cuánticos se encuentra en sus primeras etapas y debe enfrentarse aún a diversos retos entre los que se encuentra el relacionado con el desarrollo de los servicios cuánticos que, de momento, no puede realizarse de la misma forma que un servicio clásico. Por tanto, a través del enfoque de servicio híbrido, que es factible gracias al modelo *QCaaS* (*Quantum Computing as a Service*), es posible incorporar estos servicios en el desarrollo de nuevas aplicaciones de servicios que requieran de la incorporación de la computación cuántica en los procesos que implementa, denominadas aplicaciones híbridas (clásico-cuánticas).

Actualmente, existen trabajos que, mediante la computación orientada a servicios, abordan el problema de la integración [47, 55, 49], aunque esta coexistencia, actualmente, resulta bastante costosa para la parte de los servicios híbridos. Sin embargo, aún existiendo herramientas y metodologías que abordan los problemas de coste en el ámbito de los servicios clásicos, éstas no están adaptadas para tener en cuenta las características de estos nuevos servicios híbridos. Por tanto, es necesario adaptar y/o crear herramientas y metodologías que tengan en cuenta estas particularidades de los servicios cuánticos a la hora de seleccionar las máquinas donde el conjunto de servicios de las aplicaciones híbridas serán desplegados. Por ello, en este trabajo, se presenta un sistema de evaluación de coste para la obtención de un despliegue óptimo de aplicaciones híbridas. En la elección de una distribución óptima, se han tenido en cuenta dos propiedades, el coste y rendimiento, convirtiéndose en un problema multiobjetivo cuya solución, no será única, sino que serán un conjunto de soluciones que sean eficientes de Pareto, es decir, se buscará un equilibrio entre el mínimo coste y máximo rendimiento. Este conjunto de soluciones se obtiene como producto del sistema, que recibe como entrada la composición, requisitos y comportamiento de la aplicación híbrida a evaluar. Estas soluciones son resultado de la aplicación de una función de utilidad ponderada a las métricas de coste y rendimiento del conjunto de posibles configuraciones.

En cuanto a trabajos futuros, este sistema, sirve como punto de partida para continuar con la investigación del *placement problem* en las aplicaciones de servicios híbridas, ampliando no solo el análisis de nuevas variables cuantitativas que puedan afectar al comportamiento de estas aplicaciones, sino también la inclusión de nuevas máquinas distribuidas a lo largo de todo el espectro de computación continuo (Cloud-Fog-Edge), así como poder desplegar estos servicios a lo largo del Computing Continuum de forma automática según las necesidades del momento.

En relación a todo lo expuesto anteriormente, personalmente, este trabajo ha supuesto un verdadero reto, pues he tenido que hacer frente a diversos desafíos para el desarrollo de esta herramienta. Entre estos retos, cabe destacar, la formalización del estilo arquitectónico para aplicaciones de servicios híbridas. Esta formalización, ha supuesto realizar una abstracción acerca de cómo los servicios que componen una aplicación, están tanto interconectados entre sí como con las máquinas donde se despliegan. Además, de la definición de las restricciones

de dichas relaciones, teniendo en cuenta la coexistencia de servicios clásicos e híbridos y de contemplar ambos tipos de despliegues.

Por otro lado, también ha supuesto la resolución de algunos retos técnicos derivados del desarrollo del sistema en cuanto a la solución de problemas en la implementación como en la integración de diferentes herramientas.

En resumen, el desarrollo de este proyecto ha permitido alcanzar el objetivo de obtener un sistema evaluador y ha proporcionado una comprensión más profunda sobre los aspectos claves en la distribución óptima de aplicaciones híbridas. Con este trabajo, se sientan las bases para futuras investigaciones acerca de las aplicaciones cuántico-clásicas y su despliegue.

## Referencias

1. M. Salm, J. Barzen, U. Breitenbücher, F. Leymann, B. Weder, K. Wild, The nisq analyzer: automating the selection of quantum computers for quantum algorithms, in: Symposium and Summer School on Service-Oriented Computing, Springer, 2020, pp. 66–85.
2. M. Piattini, G. Peterssen, R. Pérez-Castillo, J. L. Hevia, M. A. Serrano, G. Hernández, I. G. R. de Guzmán, C. A. Paradela, M. Polo, E. Murina, et al., The talavera manifesto for quantum software engineering and programming., in: QANSWER, 2020, pp. 1–5.
3. P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM review 41 (2) (1999) 303–332.
4. L. K. Grover, A fast quantum mechanical algorithm for database search, in: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, pp. 212–219.
5. F. Gaitan, Finding flows of a navier–stokes fluid through quantum computing, npj Quantum Information 2020 6:1 6 (2020) 1–6. doi:10.1038/s41534-020-00291-0. URL <https://www.nature.com/articles/s41534-020-00291-0>
6. M. Shams, J. Choudhari, K. Reyes, S. Prentzas, A. Gapizov, A. Shehryar, M. Affaf, H. Grezenko, R. W. Gasim, S. N. Mohsin, et al., The quantum-medical nexus: Understanding the impact of quantum technologies on healthcare, Cureus 15 (10) (2023).
7. J. M. Wing, The data life cycle, Harvard Data Science Review 1 (1) (2019) 6.
8. E. Moguel, J. Rojo, D. Valencia, J. Berrocal, J. Garcia-Alonso, J. M. Murillo, Quantum service-oriented computing: current landscape and challenges, Software Quality Journal 30 (4) (2022) 983–1002.
9. A. M. Aparicio-Morales, M. Haghparast, N. Makitalo, J. Garcia-Alonso, J. Berrocal, V. Stirbu, T. Mikkonen, J. M. Murillo, Liquifying quantum-classical software-intensive system of systems, in: 2024 IEEE International Conference on Software Analysis, Evolution and Reengineering - Companion (SANER-C), 2024, pp. 159–166. doi:10.1109/SANER-C62648.2024.00028.
10. J. Garcia-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, J. M. Murillo, Quantum software as a service through a quantum api gateway, IEEE Internet Computing 26 (1) (2022) 34–41. doi:10.1109/MIC.2021.3132688.
11. M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-oriented computing: State of the art and research challenges, Computer 40 (11) (2007) 38–45.

12. J. M. Murillo, J. Garcia-Alonso, E. Moguel, J. Barzen, F. Leymann, S. Ali, T. Yue, P. Arcaini, R. Pérez, I. G. R. de Guzmán, et al., Challenges of quantum software engineering for the next decade: The road ahead, arXiv preprint arXiv:2404.06825 (2024).
13. J. Romero-Alvarez, J. Alvarado-Valiente, J. Casco-Seco, E. Moguel, J. Garcia-Alonso, J. Berrocal, J. M. Murillo, Quantum circuit scheduler for qpus usage optimization, arXiv preprint arXiv:2404.01055 (2024).
14. J. Cámara, D. Garlan, B. Schmerl, Synthesizing tradeoff spaces with quantitative guarantees for families of software systems, *Journal of Systems and Software* 152 (2019) 33–49. doi:10.1016/j.jss.2019.02.055.
15. J. L. Herrera, J. Galan-Jimenez, J. Garcia-Alonso, J. Berrocal, J. M. Murillo, Joint optimization of response time and deployment cost in next-gen iot applications, *IEEE Internet of Things Journal* 10 (2023) 3968–3981. doi:10.1109/JIOT.2022.3165646.
16. P. C. Fishburn, Utility theory, *Management science* 14 (5) (1968) 335–378.
17. A. Smith, *La riqueza de las naciones*, La Case Books, 2019.
18. A. E. Abbas, A. H. Cadenbach, On the use of utility theory in engineering design, *IEEE Systems Journal* 12 (2) (2016) 1129–1138.
19. K. Deb, *Multi-objective optimization using evolutionary algorithms*, Vol. 16, John Wiley & Sons, 2001.
20. E. M. Clarke, T. A. Henzinger, H. Veith, R. Bloem, et al., *Handbook of model checking*, Vol. 10, Springer, 2018.
21. T. Hafer, W. Thomas, Computation tree logic  $ctl^*$  and path quantifiers in the monadic theory of the binary tree, in: *International Colloquium on Automata, Languages, and Programming*, Springer, 1987, pp. 269–279.
22. M. Y. Vardi, An automata-theoretic approach to linear temporal logic, *Logics for concurrency: structure versus automata* (2005) 238–266.
23. G. Norman, D. Parker, *Quantitative verification: Formal guarantees for timeliness, reliability and performance*, Tech. rep., The London Mathematical Society and the Smith Institute (2014).
24. M. Kwiatkowska, G. Norman, D. Parker, *Stochastic model checking, Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007*, Bertinoro, Italy, May 28–June 2, 2007, *Advanced Lectures* 7 (2007) 220–270.
25. P.-A. Hsiung, Y.-R. Chen, Y.-H. Lin, Model checking safety-critical systems using safecharts, *IEEE Transactions on Computers* 56 (5) (2007) 692–705. doi:10.1109/TC.2007.1021.
26. A. Turrini, An introduction to quantum model checking, *Applied Sciences* 2022, Vol. 12, Page 2016 12 (2022) 2016. doi:10.3390/AP12042016.  
URL <https://www.mdpi.com/2076-3417/12/4/2016/html>  
<https://www.mdpi.com/2076-3417/12/4/2016>
27. R. P. Feynman, Simulating physics with computers, in: *International Journal of Theoretical Physics*, Vol. 21, Springer, 1982, pp. 467–488.
28. M. Brooks, Quantum computers: what are they good for?, *Nature* 617 (7962) (2023) S1–S3.
29. G. Bisicchia, J. Garcia-Alonso, J. M. Murillo, A. Brogi, From quantum mechanics to quantum software engineering, arXiv preprint arXiv:2404.19428 (2024).
30. K. Khan, Quantum data science: Leveraging data analytics for advancing quantum computingTFM, *Quantum Data Science* (2024). doi:10.20944/preprints202405.0495.v1.  
URL [www.preprints.org](http://www.preprints.org)

31. F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, et al., Quantum supremacy using a programmable superconducting processor, *Nature* 574 (7779) (2019) 505–510.
32. Y. Wu, W.-S. Bao, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan, et al., Strong quantum computational advantage using a superconducting quantum processor, *Physical review letters* 127 (18) (2021) 180501.
33. J. D. Hidary, J. D. Hidary, *Quantum computing: an applied approach*, Vol. 1, Springer, 2019.
34. J. E. Martinez, Decoherence and quantum error correction for quantum computing and communications, *arXiv preprint arXiv:2202.08600* (2022).
35. Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. Van Den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, et al., Evidence for the utility of quantum computing before fault tolerance, *Nature* 618 (7965) (2023) 500–505.
36. C. C. McGeoch, P. Farré, Milestones on the quantum utility highway: Quantum annealing case study, *ACM Transactions on Quantum Computing* 5 (1) (2023) 1–30.
37. J. Zhao, Quantum software engineering: Landscapes and horizons (7 2020).  
URL <https://arxiv.org/abs/2007.07047v2>
38. Quantum — digital-strategy.ec.europa.eu, <https://digital-strategy.ec.europa.eu/en/policies/quantum>, [Accessed 10-02-2024] (2024).
39. A. M. Aparicio-Morales, E. Moguel, L. M. Bibbo, A. Fernandez, J. Garcia-Alonso, J. M. Murillo, An overview of quantum software engineering in latin america, *arXiv preprint arXiv:2405.20661* (2024).
40. D. Valencia, J. Garcia-Alonso, J. Rojo, E. Moguel, J. Berrocal, J. M. Murillo, Hybrid classical-quantum software services systems: Exploration of the rough edges, in: *International Conference on the Quality of Information and Communications Technology*, Springer, 2021, pp. 225–238.
41. E. Moguel, J. Garcia-Alonso, J. M. Murillo, Development and deployment of quantum services, in: *Quantum Software: Aspects of Theory and System Design*, Springer, 2024, pp. 189–222.
42. A. M. Aparicio-Morales, J. Garcia-Alonso, J. Cámara, C. Canal, J. M. Murillo, Metodología para desplegar aplicaciones híbridas (cuántico-clásicas), *Actas de las XIX Jornadas de Ciencia e Ingeniería de Servicios (JCIS 2024)*.  
URL <https://hdl.handle.net/11705/JCIS/2024/34>
43. A. Gamez-Diaz, P. Fernandez, A. Ruiz-Cortes, Automating sla-driven api development with sla4oai, in: *International Conference on Service-Oriented Computing*, Springer, 2019, pp. 20–35.  
URL [https://doir.org/10.1007/978-3-030-33702-5\\_2](https://doir.org/10.1007/978-3-030-33702-5_2)
44. M. Kang, D. I. Kang, J. P. Walters, S. P. Crago, A comparison of system performance on a private openstack cloud and amazon ec2, *IEEE International Conference on Cloud Computing, CLOUD 2017-June (2017)* 310–317, expone aproximadamente el rendimiento de las máquinas de low to moderate. doi:10.1109/CLOUD.2017.47.
45. S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, R. Duncan, t—ket<sub>2</sub>: a retargetable compiler for nisq devices, *Quantum Science and Technology* 6 (1) (2020) 014003.
46. A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, M. Martonosi, Scaffcc: A framework for compilation and analysis of quantum computing programs, in: *Proceedings of the 11th ACM Conference on Computing Frontiers*, 2014, pp. 1–10.



47. J. Romero-Álvarez, J. Alvarado-Valiente, E. Moguel, J. García-Alonso, J. M. Murillo, Using open api for the development of hybrid classical-quantum services, in: International Conference on Service-Oriented Computing, Springer, 2022, pp. 364–368.
48. J. Alvarado-Valiente, J. Romero-Álvarez, J. Garcia-Alonso, J. M. Murillo, A guide for quantum web services deployment, in: International Conference on Web Engineering, Springer, 2022, pp. 493–496.
49. H. T. Nguyen, M. Usman, R. Buyya, Qfaas: A serverless function-as-a-service framework for quantum computing, *Future Generation Computer Systems* 154 (2024) 281–300.
50. Y. Hu, C. de Laat, Z. Zhao, Optimizing service placement for microservice architecture in clouds, *Applied Sciences* 2019, Vol. 9, Page 4663 9 (2019) 4663, importante. doi:10.3390/APP9214663.  
URL <https://www.mdpi.com/2076-3417/9/21/4663/html>  
<https://www.mdpi.com/2076-3417/9/21/4663>
51. M. Selimi, L. Cerda-Alabern, M. Sanchez-Artigas, F. Freitag, L. Veiga, Practical service placement approach for microservices architecture, *Proceedings - 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CC-GRID 2017* (2017) 401–410Este nos puede servir como herramienta diferenciante en cuanto a cómo obtener una ubicación óptima ya que utiliza el estado de la red para seleccionar el computador (mas o menos) y esto puede ser un horizonte para ampliar la herramienta con la selección del computador que tenga menos cola en qiskit. doi:10.1109/CCGRID.2017.28.
52. J. Cámara, D. Garlan, B. Schmerl, Synthesizing tradeoff spaces with quantitative guarantees for families of software systems, *Journal of Systems and Software* 152 (2019) 33–49. doi:10.1016/J.JSS.2019.02.055.
53. S. Khan, M. Farnsworth, R. McWilliam, J. Erkoyuncu, On the requirements of digital twin-driven autonomous maintenance, *Annual Reviews in Control* 50 (2020) 13–28.
54. A. Bayerstadler, G. Becquin, J. Binder, T. Botter, H. Ehm, T. Ehmer, M. Erdmann, N. Gaus, P. Harbach, M. Hess, et al., Industry quantum computing applications, *EPJ Quantum Technology* 8 (1) (2021) 25.
55. J. Alvarado-Valiente, J. Romero-Álvarez, E. Moguel, J. García-Alonso, J. M. Murillo, Technological diversity of quantum computing providers: a comparative study and a proposal for api gateway integration, *Software Quality Journal* 32 (1) (2024) 53–73.

## Anexo A. Estilo Arquitectónico Aplicaciones Híbridas (Cuántico-Clásicas)

ModelType: dtmc;

```

abstract sig PU {services : some Service}
</
formula cpulogicalperformancefactor;
formula cpuram;
formula cpubandwidth;
formula cpucostfactor;
formula qpuprize;
/>
abstract sig QPU extends PU {}
</
formula cpulogicalperformancefactor;
formula cpuram;
formula cpubandwidth;
formula cpucostfactor;
formula qpuprize;
var dummy : bool init false
[services:qpuprize] true -> (dummy'=true);
/>
abstract sig CPU extends PU {}
</
formula cpulogicalperformancefactor;
formula cpuram;
formula cpubandwidth;
formula cpucostfactor;
formula qpuprize;
var dummy : bool init false
[services:cpuram] true -> (dummy'=true);
/>
abstract sig Service {
    machines : some PU,
    deployment : one Deployment,
    hybrid_service: set Service, // Simular el
        concepto de hybrid_service
    link: some Service // Comunicacion entre
        servicios del caso de uso
}
</
formula mslogicalperformancefactor;
formula msram;
formula msbandwidth;

```

```

        formula msexecutiontime;
        formula msavailability;
        formula shots;
    />
    abstract sig ClassicalService extends Service {}
    </
        formula mslogicalperformancefactor;
        formula msram;
        formula msbandwidth;
        formula msexecutiontime;
        formula msavailability;
        formula shots;
    />
    abstract sig QuantumService extends Service {}
    </
        formula mslogicalperformancefactor;
        formula msram;
        formula msbandwidth;
        formula msexecutiontime;
        formula msavailability;
        formula shots;
    />
    abstract sig Deployment {services: some Service}
    </
        formula dummy;
    />
    abstract sig HybridDeployment extends Deployment {}
    </
        formula dummy;
        var finished : bool init false;
        [] (finished=false) -> (finished'=true);
    />
    abstract sig ClassicalDeployment extends Deployment {}
    </
        formula dummy;
        var finished : bool init false;
        [] (finished=false) -> (finished'=true);
    />
    lone sig HybridUseCase extends HybridDeployment {}
    </
        formula dummy = 0;
    />
    lone sig ClassicalUseCase extends ClassicalDeployment {}
    </
        formula dummy = 0;

```

```

/>

fact {
//————Architectural—Restrictions —————
//————Machines— Restrictions—————
// Toda maquina debe estar asociada a algun servicio
all pu: PU | #pu.services > 0
// Si un servicio esta desplegado en una maquina,
// entonces esa maquina solo puede estar relacionada con
// ese servicio
all pu: PU, s: Service | s in pu.services implies s.
machines = pu
// Para todas las maquinas classicas , su conjunto de
// servicios no pueden ser cuanticos
all c: CPU | #(c.services & QuantumService) = 0
// Todos los servicios deben de estar asociados a un
// despliegue
// Para todas las maquinas classicas , su conjunto de
// servicios no pueden ser cuanticos
all q: QPU | #(q.services & ClassicalService) = 0
//————Services— Restrictions—————
//all s: Service | #(s.deployment) = 1 // Ya se cumple
all s: Service | #(s.deployment & HybridDeployment) = 0
// implies #(s.hybrid_service) = 0
// Todo servicio tiene que estar asociado a un despliegue
// . Aqui se define para ambos lados, desde servicios y
// desde despliegue
all s: Service, d: Deployment | (s in d.services implies
d in s.deployment) and (s.deployment = d implies s in
d.services)
// Ningun servicio puede estar relacionado consigo mismo
all s: Service | s not in s.link
//————Classical—Restrictions—————
// Un servicio clasico , no puede relacionarse con una
// maquina cuantica
all cs: ClassicalService | #(cs.machines & QPU) = 0
// Un servicio clasico , no puede estar alojado en una
// misma instancia de maquina clasica que otro servicio
// clasico (No co-alojados)
all cs1, cs2 : ClassicalService | cs1 != cs2 implies #(
cs1.machines & cs2.machines) = 0
// Un servicio clasico , no puede estar relacionado con
// mas de una instancia de maquina clasica.
all sc: ClassicalService | #sc.machines < 2

```

```

// Un servicio clasico , tiene que tener una instancia de
// maquina clasica asociada.
all sc: ClassicalService | #sc.machines > 0
//-----Quantum-Restrictions-----
// Un servicio cuantico , no puede estar alojado en una
// maquina clasica
all qs: QuantumService | #(qs.machines & CPU) = 0
// Un servicio cuantico , no puede estar alojado en mas de
// una maquina cuantica
all qs: QuantumService | #qs.machines < 2
//Un servicio cuantico , tiene que tener una instancia de
// maquina cuantica asociada.
all qs: QuantumService | #qs.machines > 0
// Toda maquina cuantica debe tener un servicio asociado
all qs: QuantumService, q: QPU | qs.machines = q implies
// qs in q.services
// Todo servicio cuantico debe tener un servicio clasico
// asociado
all qs: QuantumService | #(qs.hybrid_service) = 1 and #(
// qs.hybrid_service & QuantumService) = 0
//-----Hybrid-Services-Restrictions-----
// Todo servicio cuantico tiene que estar asociado a un
// servicio clasico
all qs: QuantumService | qs in ClassicalService.
// hybrid_service
// Para todos los servicios clasicos y cuanticos que
// conforman un servicio hibrido , deben pertenecer al
// mismo despliegue
all s1, s2: Service | s1 !=s2 and s1 in s2.hybrid_service
// and s2 in s1.hybrid_service implies s1.deployment =
// s2.deployment
// Un servicio cuantico no puede compartir el servicio
// clasico con el que esta asociado
all qs1, qs2: QuantumService | qs1 != qs2 and #(qs1.
// hybrid_service) = 1 and #(qs2.hybrid_service) = 1
// implies #(qs1.hybrid_service & qs2.hybrid_service) = 0
// El servicio cuantico y clasico que conformen un
// servicio hibrido , solo pueden estar relacionado entre
// ellos
all qs:QuantumService, cs: ClassicalService | (cs in qs.
// hybrid_service implies qs in cs.hybrid_service) and (
// qs in cs.hybrid_service implies cs in qs.
// hybrid_service)

```

```

// Dos servicios clasicos no pueden formar un servicio
// hibrido. Un servicio consigo mismo tampoco puede
// formar un servicio hibrido
all cs1, cs2: ClassicalService, hd: HybridDeployment | (
    cs1 in hd.services and cs2 in hd.services) implies not
    (cs1 in cs2.hybrid_service) and not ( cs2 in cs2.
    hybrid_service)
// Dos servicios forman un servicio hibrido, entonces
// esos dos estan relacionados a traves de link
all s1, s2: Service | s1 !=s2 and s1 in s2.hybrid_service
    and s2 in s1.hybrid_service implies (s2 in s1.link
    and s1 in s2.link)
//-----Deployment-Restrictions-----
// Todos los despliegues hibridos tienen al menos un
// servicio cuantico y al menos un servicio clasico
all hd: HybridDeployment | #(hd.services & QuantumService
    ) > 0 and #(hd.services & ClassicalService) > 0
// Todo despliegue clasico no tiene servicios cuanticos
all cd: ClassicalDeployment | #(cd.services &
    QuantumService) = 0
//-----
}

```