

Generación de Código Clásico-Cuántico desde modelos UML

¹Ricardo Pérez-Castillo^[0000-0002-9271-3184], ¹Luis Jiménez-Navajas^[0000-0001-6257-7153], ¹Iván Cantalejo y ²Mario Piattini^[0000-0002-7212-8279]

¹ Facultad de Ciencias Sociales y Tecnologías de la Información,
Universidad de Castilla-La Mancha, Talavera de la Reina 45600, España

² Escuela Superior de Informática,
Universidad de Castilla-La Mancha, Ciudad Real 13071, España
ricardo.pdelcastillo@uclm.es, luis.jimeneznavajas@uclm.es,
ivan.cantalejo@alu.uclm.es, mario.piattini@uclm.es,

Resumen. La computación cuántica ha ganado mucho interés en los últimos años, gracias a su capacidad de resolver problemas prácticos que la computación clásica no puede abordar. Estas aplicaciones se conseguirán mediante el desarrollo de software cuántico, por lo que muchas organizaciones están iniciando proyectos para integrar este nuevo software en sus sistemas. Existen procesos de modernización software (basados en la ingeniería dirigida por modelos) para migrar desde/hacia los denominados sistemas software híbridos, que integran software clásico y cuántico. En este proceso ya se han abordado las fases de ingeniería inversa y reestructuración, pero aún no se ha abordado la fase de ingeniería directa para estos sistemas. Este artículo presenta una técnica de generación de código desde modelos de diseño UML extendidos con un perfil cuántico. Se propone una transformación EGL que genera código Python y Qiskit, integrando así el código clásico y cuántico. La transformación se ha validado con éxito, de forma preliminar, con el diseño de un sistema híbrido para una aplicación financiera. Este trabajo completa el proceso de modernización cuántica, lo que facilitará la adopción de este nuevo paradigma en la industria.

Palabras Clave: Software Cuántico, Generación de Código, MDE, UML, EGL

1 Introducción

La computación cuántica está destinada a revolucionar la forma en que la sociedad resolverá ciertos problemas que suponen un desafío computacional para los ordenadores clásicos. Es decir, aquellos problemas que no se pueden abordar en un marco de tiempo razonable, incluso utilizando el supercomputador clásico más grande del mundo. Por lo tanto, la computación cuántica está ganando cada vez más relevancia debido a sus aplicaciones potenciales en varios campos y sectores empresariales [1-3].

En los últimos años, el interés en la Computación Cuántica ha aumentado drásticamente. La década actual (2020) es la "década cuántica", en la que "la computación cuántica está preparada para expandir el alcance y la complejidad de los problemas

comerciales que pueden ser resueltos” [2] ofreciendo una verdadera “ventaja cuántica”. El potencial de la computación cuántica es inmenso en casi todos los sectores productivos: servicios financieros [4] y economía [5], simulación química y diseño molecular [6], la medicina y desarrollo de medicamentos [7], seguridad de la información [8], cadenas de suministro y logística [9], o la inteligencia artificial [10, 11], etc. La computación cuántica se encuentra en un punto de inflexión con barreras significativas que por cruzar, pero un mundo de oportunidades por delante [12]. Esto significa que las empresas deben estar preparadas para esta nueva transición tecnológica.

La computación cuántica se basa en los principios contraintuitivos de la mecánica cuántica, como la superposición y el entrelazamiento. La unidad computacional de las computadoras cuánticas se llama cúbit, que pueden representarse con diferentes sistemas físicos (por ejemplo, el espín del electrón, la polarización de un solo fotón, etc.) [13]. Actualmente es posible distinguir dos paradigmas principales de computación cuántica: basada en puertas cuánticas y adiabática. La computación basada en puertas también se conoce como computación cuántica universal (ya que es *Turing-completo*) y es en la que se centra la presente investigación.

La supremacía cuántica ya ha sido demostrada [14] y constantemente aparecen computadoras cuánticas más potentes. Google ha predicho máquinas de 1 millón de cúbits antes del final de la década [15].

A pesar de la demostración preliminar de muchos de estos avances y su potencial, las ventajas que ofrece la computación cuántica no se pueden lograr mediante el uso exclusivo de computadoras cuánticas de forma aislada, sino que también se requiere del software cuántico, lo cual sin duda desempeñará un papel importante [16, 17].

Las técnicas de programación de software cuántico que tenemos hoy en día se han propuesto experimentalmente de una forma, mayoritariamente, *ad hoc*. En consecuencia, la Ingeniería del Software Cuántico (QSE) [18-21] ha surgido como un nuevo campo de investigación dirigido a producir software cuántico mediante la aplicación de los conocimientos y lecciones aprendidas del campo de la ingeniería de software clásico. Esto requiere la aplicación o adaptación de los procesos, métodos, técnicas, prácticas y principios de ingeniería de software existentes para el desarrollo de software cuántico, así como la creación de otros nuevos.

La ingeniería del software cuántica puede ayudar a resolver una de las necesidades pendientes: cómo migrar hacia, y desde, los denominados sistemas software híbridos [19, 22]. Esto es un desafío ya que las empresas no descartarán sus sistemas de información clásicos porque soportan los procesos de negocio críticos. En lugar de una sustitución completa, los sistemas de información clásicos y cuánticos operarán juntos en estos sistemas software híbridos [22]. Estos sistemas soportan funcionalidad de negocio implementadas con lenguajes de programación clásicos (*driver*) que realizan llamadas a software cuántico ejecutado en la nube en ordenadores cuánticos.

Una estrategia para la migración de/hacia sistemas software híbridos es la modernización del software. Este proceso se basa en la reingeniería tradicional, pero sigue los principios de la Ingeniería Dirigida por Modelos (MDE). La modernización de software consta de tres fases: (i) ingeniería inversa, (ii) reestructuración; y (iii) ingeniería directa. Ya existen algunas propuestas en las fases de ingeniería inversa [23, 24], así como en

la de reestructuración [25]. Sin embargo, la generación de código desde modelos en la fase de ingeniería directa no ha sido abordada aún.

Aunque existen muchas propuestas de generación de código cuántico desde diferentes tipos de modelos, no se tiene en cuenta la generación conjunta de software clásico y cuántico desde modelos de diseño de alto nivel que consideran relaciones entre ambos tipos de paradigma. Esta es la mayor contribución de esta investigación. En concreto, se propone una técnica generativa basada en una transformación *model-to-text* que es capaz de generar código Python y Qiskit desde modelos UML extendidos (que consideran conjuntamente software clásico y cuántico). La transformación se ha implementado en EGL (*Epsilon Generation Language*) y se ha integrado en una herramienta experimental. Esta implementación ha permitido realizar una validación preliminar con un diseño UML de un pequeño sistema financiero que hace uso de varias funcionalidades cuánticas. La principal implicación de este trabajo es que cubre la fase de ingeniería directa, lo que permite completar el proceso de modernización de sistemas software híbridos. Pensamos, que esto contribuirá a facilitar la adopción del software cuántico en la industria.

El resto del artículo se estructura de la siguiente forma. La sección 2 explica brevemente el estado del arte. La sección 3 presenta los detalles de la técnica de generación de código propuesta. La sección 4 presenta una evaluación preliminar a través de su aplicación con el diseño UML de un sistema software híbrido. Finalmente, la sección 5 presenta las conclusiones y trabajo futuro.

2 Estado del Arte

En esta sección se introduce brevemente la ingeniería del software cuántica (sección 2.1), así como la modernización de sistemas software híbridos y trabajos relacionados en este campo (sección 2.2).

2.1 Software Cuántico

Los avances en la computación cuántica se han producido simultáneamente con el desarrollo de los lenguajes de programación cuántica y sus respectivos compiladores [17]. Hay una gran variedad de lenguajes de Programación cuánticos [26, 27]: OpenQASM, Q#, diferentes extensiones para Python, por ejemplo, Qiskit (IBM), Cirq (Google) o pyQuil (Rigetti), entre muchos otros.

La combinación de física cuántica e informática cambia la forma en que los programas se diseñan, desarrollan y ejecutan [28]. Mientras que los programas clásicos calculan resultados únicos de forma determinista, los programas cuánticos se diseñan y codifican para explorar un rango reducido de posibilidades después de explorar el espacio de búsqueda de una manera probabilística y en paralelo. La mayoría de los programas cuánticos actuales se ejecutan de forma remota en computadoras cuánticas en la nube, y los resultados que devuelven son interpretados por programas clásicos.

Las diferencias radicales entre el software cuántico y el clásico han llevado a matemáticos y científicos a seguir una forma no sistemática de codificar el software cuántico

[18]. Sin embargo, con la ‘industrialización’ del software cuántico y su creciente uso en varios dominios comerciales, ha hecho que exista una mayor demanda de producir software cuántico de forma más sistemática y asegurando su calidad, tal y como se afirma en el Manifiesto de Talavera para la Programación e Ingeniería de Software Cuántico [16]. Así, el campo de la ingeniería del software cuántico (QSE) [20, 21] comenzó a desarrollarse de forma incipiente, tomando buenas prácticas y lecciones aprendidas de la ingeniería del software clásica.

2.2 Modernización de Sistemas Software Híbridos

En general, los sistemas software híbridos enfrentan obstáculos en la portabilidad del código, la integración de herramientas, la validación y verificación de programas, y la orquestación del flujo de trabajo [29]. Otro desafío importante es la modernización de dichos sistemas híbridos siguiendo las buenas prácticas de ingeniería de software, como son la automatización, la facilidad de comprensión, la abstracción de diseños de alto nivel [25], la preservación del conocimiento empresarial integrado [23], aseguramiento de la calidad [30], software orientado a servicios [31], etc. En este sentido, la modernización del software se ha adaptado [22] para abordar los problemas asociados con las migraciones de estos sistemas software híbridos.

La modernización del software se basa en la reingeniería tradicional, pero sigue los principios de la Ingeniería Dirigida por Modelos (MDE). La modernización de software consta de tres fases: (i) ingeniería inversa, para inspeccionar y abstraer información en un nivel de abstracción superior; (ii) reestructuración, para mejorar el diseño interno y agregar nuevas funcionalidades al más alto nivel de abstracción; y (iii) ingeniería directa, para generar el código fuente de destino. Además, en esta investigación en concreto, se propone una modernización de software basada en estándares existentes como UML (*Unified Modelling Language*) y KDM (*Knowledge Discovery Metamodel*) (véase Fig. 1). La modernización de software híbrido puede ser utilizada en al menos tres escenarios no excluyentes: (i) migrar los sistemas clásicos de información heredados hacia arquitecturas híbridas que apoyan la integración de software clásico-cuántico; (ii) migrar algoritmos cuánticos aislados existentes e integración en los sistemas híbridos (altamente automatizable); y (iii) transformar o agregar nuevas funcionalidades de negocio soportadas por software cuántico, que se integrarán en los sistemas híbridos resultantes. Cualquiera de estos escenarios, requerirán en su última fase, generar código de forma (semi)automática desde los modelos de diseño abstractos. Es precisamente esta fase en la que se centra la investigación presente.

3 Generación de Código Híbrido desde modelos UML extendidos

La propuesta principal de este artículo es una técnica de generación de código para sistemas software híbridos desde modelos de diseño abstractos (véase Fig. 2). En cuanto a la entrada, se proponen modelos UML extendidos mediante un perfil para software cuántico [32].

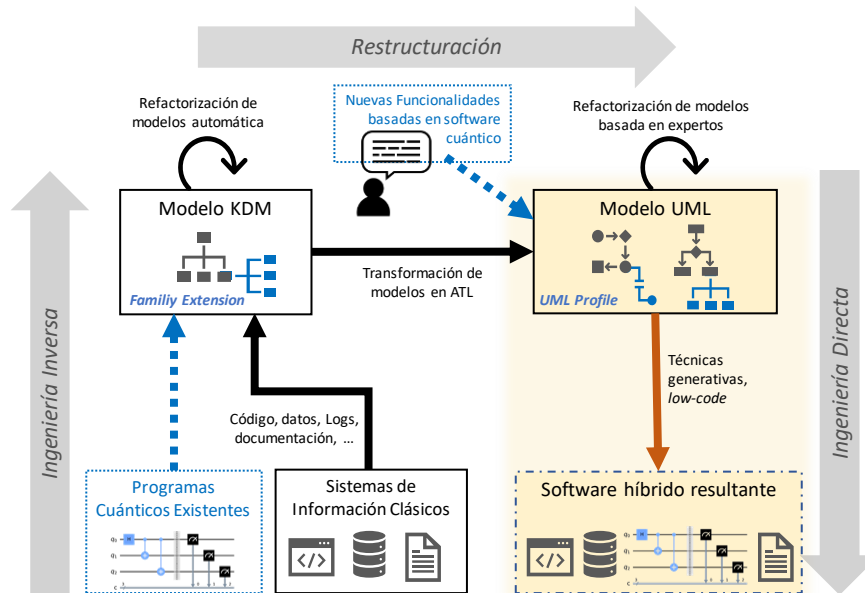


Fig. 1. Proceso de modernización de sistemas software híbridos

En cuanto a la salida, se propone que el código fuente destino sea Python y su extensión de software cuántico Qiskit [33] de IBM, ya que en la actualidad ambos lenguajes son ampliamente utilizados, tanto en desarrollo de software clásico, como en su contraparte cuántica. Además, Qiskit da soporte a la generación de código cuántico ensamblador OpenQASM, que es la base de otros muchos lenguajes de programación cuánticos. A pesar de esto, la transformación se podría adaptar a otros modelos y lenguajes de programación dentro del enfoque MDE.

En cuanto al proceso general de la transformación descrito en la Fig. 2, cabe destacar que la transformación se lleva a cabo en dos pasos (que incluso pueden verse como dos transformaciones independientes). El código EGL de las transformaciones está disponible en [34].

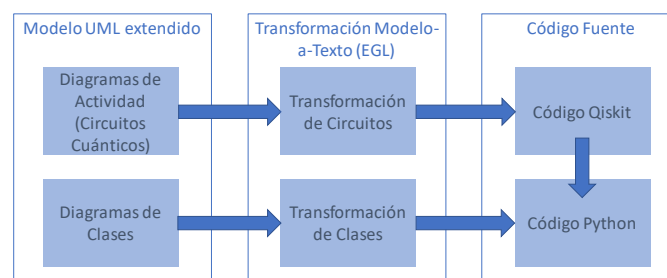


Fig. 2. Proceso general de transformación UML a código fuente Python.

- En un primer paso (parte superior) se genera el software cuántico en Qiskit, desde los circuitos cuánticos (véase la sección 3.2) que vienen representados en diagramas de actividad extendidos de acuerdo con el perfil UML (véase la sección 3.1).
- En un segundo paso (parte inferior) se genera el código Python clásico desde el diagrama de clases UML (véase la sección 3.3). Estos diagramas de clases usan también estereotipos del perfil cuántico de UML que permiten, entre otras cosas, definir que algoritmo (circuito) cuántico es usado dese ciertas partes del software clásico. Esta información es sumamente útil para establecer dependencias de uso en el código fuente resultante, entre la parte clásica y cuántica. Por este motivo, la transformación de circuitos cuánticos se debe realizar en primer lugar.

3.1 Perfil de UML para software cuántico

El perfil de UML para software cuántico está fuera del alcance de esta propuesta. No obstante, en aras de comprender el modelo de entrada de la transformación propuesta, esta sección introduce brevemente este perfil, que se explica de forma detallada en un trabajo previo [32]. Este perfil surge de la necesidad de diseñar y construir software cuántico de forma sistemática [35]. Las descripciones abstractas de software son clave para (i) discutir entre ingenieros y desarrolladores las preocupaciones de diseño, y (ii) modelar sistemas con representaciones conceptuales de alto nivel mientras los detalles de implementación de bajo nivel permanecen ocultos. Debido a esta necesidad, se escogió el Lenguaje de Modelado Unificado (UML) [36], dado que aporta una solución contrastada y ampliamente utilizada en la industria del software clásico.

El perfil UML cuántico cubre tanto aspectos estructurales como del comportamiento, es decir, la parte estática y dinámica de un sistema de información. En concreto, la extensión UML propuesta aborda los diagramas de casos de uso, clase, secuencia, actividad y despliegue. Como resultado, tanto los elementos cuánticos como los clásicos, así como las relaciones entre ellos, se pueden modelar de forma conjunta en un diseño integrado a través de UML, lo cual es clave en el proceso de modernización software que se aborda en este trabajo. Aunque en propuestas existentes previas UML se ha extendido a través de diversos Lenguajes Específicos de Dominio (DSL) [37], esta propuesta extiende UML a través de un perfil que es el mecanismo estándar para este fin. Aunque en comparación con los DSL, la expresividad de una extensión cuántica implementada mediante un perfil UML es limitada, la principal ventaja de esta estrategia de extensibilidad es que los modelos de diseño resultantes siguen siendo compatibles con UML [38]. En consecuencia, no es necesario adaptar ni las herramientas UML existentes ni entrenar a los diseñadores de software, ya que la notación es bien conocida y está respaldada por la mayoría de las herramientas UML del mercado.

En concreto, la transformación propuesta considera dos tipos de diagramas: (i) el diagrama de clases (véase ejemplo en la Fig. 3), que proporciona la vista estática integral del sistema, representando como se organizan las clases y paquetes; y (ii) el diagrama de actividades (véase ejemplo en la Fig. 4) que sirve para modelar la dinámica (algoritmos o circuitos cuánticos) de ciertos componentes software cuánticos. Estos dos diagramas han sido utilizados como ejemplo para validar la transformación EGL propuesta (véase sección 4).

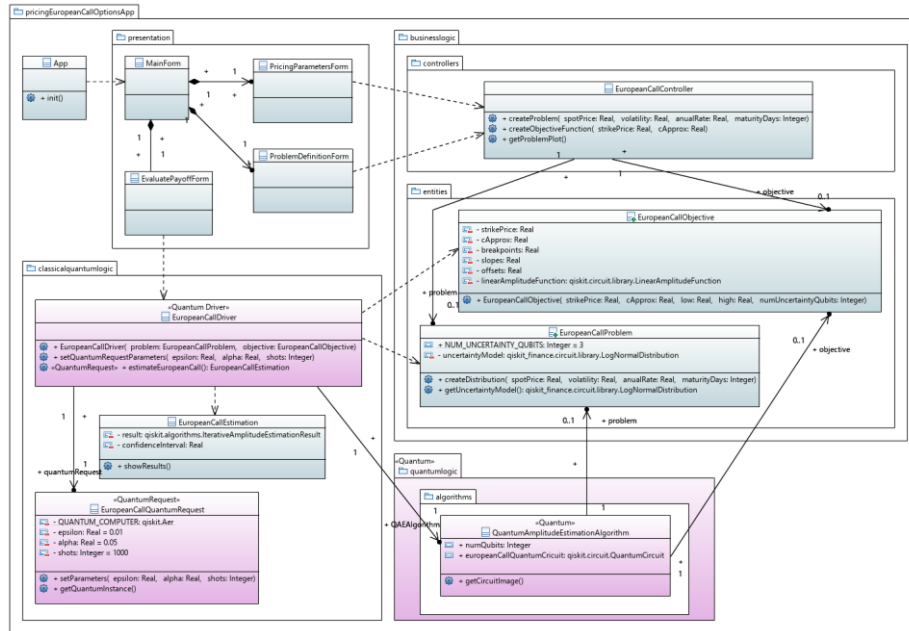


Fig. 3. Diagrama de clases usando el perfil de UML para software cuántico (adaptado de [32]).

Desde el diagrama de clases, se puede establecer relaciones entre una clase, estereotipada con <<Quantum>> y el código cuántico correspondiente que se modela con el diagrama de actividad correspondiente. La Tabla 1 resume los estereotipos disponibles en el perfil UML que se pueden aplicar a estos dos tipos de diagramas.

3.2 Transformación de Diagramas de Actividad a código Qiskit

Esta primera transformación EGL está encaminada a generar el código cuántico (Qiskit) de los circuitos cuánticos representados mediante el diagrama de actividad UML extendido. En concreto, se genera un fichero Python (.py) importando todos los paquetes necesarios de Qiskit, y generando un objeto de la clase *QuantumCircuit* donde se modelará el circuito cuántico. En cada circuito se incorporan los elementos necesarios en el orden siguiente:

- Se generan los registros cuánticos utilizando la clase *QuantumRegister* que generan agrupación de cúbits. De acuerdo con el perfil cuántico UML, los elementos ‘Lane’ en el diagrama de actividad representan un cúbit, por lo que se generará un registro de cúbits con el mismo número de elementos ‘Lane’.
- De forma similar se genera un registro de variables clásicas (*ClassicalRegister*) donde se almacenan los valores de las mediciones. Esto se realiza de acuerdo con las puertas de medición (*measure*) que vienen representadas en el modelo UML como elementos ‘Action’ estereotipados con <<Measure>>.



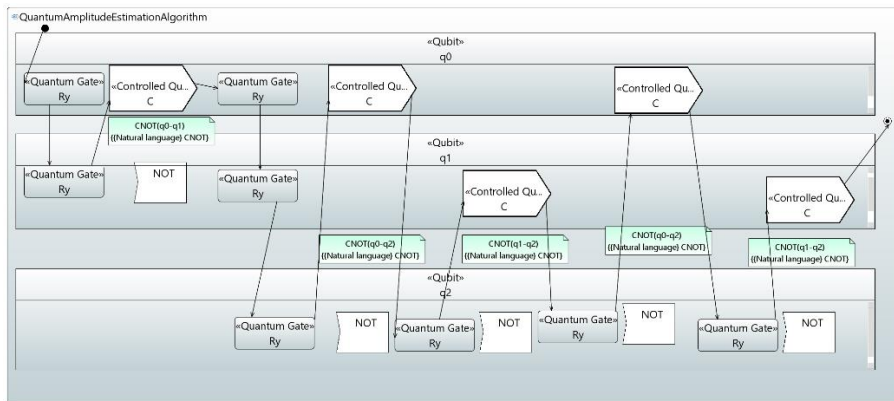


Fig. 4. Ejemplo de diagrama de actividad usando el perfil que representa un circuito cuántico.

Tabla 1. Estereotipos definidos en el perfil UML en relación con los elementos UML

Stereotype	Class						Activity				
	Package	Class	Association Class	Dependency	Operation	Swimlane	Activity	Activity Partition	Action	Accept Event Action	Send Signal Action
Quantum	•	•									
Quantum Request		•	•	•	•						
Quantum Driver		•									
Quantum Algorithm						•					
Quantum Circuit							•				
Qubit								•			
Quantum Gate									•	•	
Measure									•		
Reset									•		
Controlled Qubit											•

- Se instancia el circuito cuántico en Qiskit con los dos registros previos, como información mínima necesaria para crearlo.
- Se añaden al circuito cuántico las puertas cuánticas que se van aplicando sobre los diferentes cúbits, así como las operaciones de medición. Las puertas cuánticas se modelan como elementos ‘Action’ estereotipados con <<Quantum Gate>>. Es clave respetar el orden de aplicación de las puertas, ya que de alguna forma los circuitos cuánticos (también en diagrama de actividad) no define de forma explícita el flujo de control y el orden de ejecución de las puertas (más allá de restricciones por algunas dependencias en el orden) (véase Fig. 5).
- Otra cuestión importante son las puertas cuánticas controladas, que en el diagrama de actividad se modelan como una pareja de elementos más una restricción entre ambos: (i) ‘SendSignalAction’ (sobre el cúbit que sirve para hacer el control); y (ii) ‘AcceptEventAction’ (sobre la que se aplica la operación de la puerta cuántica). En este caso, la Transformación EGL debe tener en cuenta la relación entre estos dos



elementos en diferentes elementos ‘Lane’ (cúbits) en el circuito, a la vez que se preserva el orden de aplicación respecto de las otras puertas cuánticas.

- Finalmente, se devuelve el circuito y se realiza la persistencia del fichero Python.

```

1  [*QUANTUM GATES*]
2  [% var initial = getInitialNode();
3     if(initial == null){
4         "There is no InitialNode".println();
5     } else{
6         var actual = initial.outgoings.target.get(0);
7         var type = actual.toString().split(' ').get(0).split('@').get(0);
8         while(not(type == "org.eclipse.uml2.uml.internal.impl.ActivityFinalNodeImpl")){
9             var gate = checkGateType(actual);
10            if (not (gate == "pass")) {[%
11                [%=circuitName%].[%=gate%]
12                [%]
13                actual = nextNode(actual);
14                type = actual.toString().split(' ').get(0).split('@').get(0);
15            }
16        }
    ]%]

```

Fig. 5. Extracto de la transformación EGL para la generación de puertas cuánticas en Qiskit

3.3 Transformación de Diagramas de Clase a código Python

En un segundo paso se lleva a cabo la transformación del diagrama de clases UML a código Python. Ya existen varias herramientas generativas que transforman diagramas de clases UML en diferentes lenguajes de programación. En este sentido, la transformación propuesta sigue los mismos principios y mapeo entre elementos:

- Se definen los paquetes (así como la correspondiente estructura de carpetas para generar los diferentes archivos .py resultantes) a partir de los paquetes y subpaquetes del diagrama de clases.
- Se genera un fichero .py por cada clase en el diagrama.
- Para cada clase, se definen los atributos en código. Es necesario establecer un mapeo entre los tipos de datos predefinidos en UML y los tipos de datos en Python. Esto es parametrizable para que la transformación pueda ser extendida a otros lenguajes de programación. Junto con los atributos se definen los métodos de la interfaz pública ‘getters’ y ‘setters’.
- Finalmente, las operaciones definidas en las clases del diagrama UML se definen como funciones en código Python en la clase correspondiente. Evidentemente, estas operaciones están vacías y deben ser completadas, pero la signatura de la función con los parámetros y el tipo de retorno es creada en la clase resultante.

Además de estas reglas de transformación, la novedad de esta propuesta reside en que se considera la información del perfil cuántico y es capaz de integrar este código con los circuitos cuánticos generados desde los diagramas de actividad en el paso anterior. Para realizar la integración se reconoce las siguientes condiciones tanto en el diagrama de clases UML como en los ficheros Python generados para los diagramas de actividad:

- Existe una clase está estereotipada como <<Quantum Driver>>

- Desde la anterior, existe una dependencia hacia otra clase, la cual está a su vez estereotipada como <<Quantum>>.
- El nombre de la clase estereotipada como <<Quantum>> coincide con el nombre de un circuito cuántico (archivo .py) generado previamente desde un diagrama de actividad.

Bajo esas condiciones, en el fichero de código resultante para la clase 'driver', se genera un atributo (si es que no fue ya explícitamente definido en la clase UML) de tipo 'qiskit.circuit.QuantumCircuit' y con el nombre de la otra clase estereotipada con <<Quantum>> y cuyo código fue previamente generado (véase Fig. 6).

```

1 [%function getAttributeType(attribute) : String{
2     if(attribute.type.name == null)
3         return getDataTypes(attribute.type.eProxyURI.toString().split('#').get(1));
4     else if(attribute.type.name == "qiskit.circuit.QuantumCircuit")
5         for(activity in activities.replace(' ', '').split(','))
6             if(not(attribute.name == null))
7                 if((activity.replace(' ', '_').replace('-', '_')) ==
8                     (attribute.name.replace(' ', '_').replace('-', '_').toLowerCase()))
9                     return attribute.name;
10    return getDataTypes(attribute.type.name);
11 }

```

Fig. 6. Extracto de la función EGL para la integración de código clásico-cuántico en Python

3.4 Ejecución de la transformación

La transformación EGL puede ser ejecutada directamente mediante Papyrus (Eclipse), que sigue un enfoque MDE y permite la visualización de los modelos UML. No obstante, se está desarrollando una pequeña aplicación en *JavaFX* que permite cargar los modelos UML y definir la salida donde se generará el código Python. En cualquier caso, la puesta en producción y mejora de esta herramienta es un trabajo en curso.

4 Ejemplo de Aplicación

Los ejemplos de modelos UML presentados anteriormente en Fig. 3 y Fig. 4 han sido utilizados para probar la transformación de modelo UML a código Python que se propone. Estos ejemplos se basan en la aplicación desarrollada previamente en [32] y que se encuentra disponible en [39]. Este es un ejemplo real de un sistema software híbrido que ha sido diseñado con un modelo UML extendido con el perfil presentado. En particular se modela una aplicación financiera que dado un contrato de opción de compra de un activo a un precio predeterminado antes o al día de vencimiento, debe estimar el pago, es decir, el precio de la opción y el análisis de riesgo. Para el diseño del sistema híbrido se ha considerado una arquitectura multicapa, y se ha orientado a ser construido como una aplicación web. Mientras, el algoritmo de optimización se ha diseñado como un circuito de estimación de amplitud cuántica, que aplica la inversa de la transformada de Fourier (QFT).

Utilizando el diseño en UML mencionado, se desarrolló la aplicación web en *Flask* (Python) que puede probarse en [40]. Dado que ya existe el código desarrollado

previamente de forma manual, y tras aplicar la transformación EGL, se pudo comparar el código generado [34] con el existente [39]. Tras comparar el código, se observó que ambos artefactos coincidían en un alto grado, exceptuando algunos aspectos:

- El código generado define solo el esqueleto de las clases clásicas, mientras el existente presenta código totalmente funcional. En cualquier caso, este no es el objetivo de esta técnica de generación automática de código.
- El código generado del circuito cuántico varía en el orden de aplicación de algunas puertas, pero solo en los casos en que no afecta al resultado final, por lo que se pueden considerar circuitos funcionalmente equivalentes.
- Otras excepciones menores se debieron a que los desarrolladores que codificaron de forma manual el sistema se tomaron, probablemente, algunas licencias respecto al diseño UML original (que suele ocurrir en entornos reales). No obstante, esto no se puede considerar un error en la transformación propuesta.

A pesar de estas excepciones, se demostró que la transformación es capaz de generar código totalmente compilable con Python y Qiskit, y que además lo hace de forma estructura en paquetes y clases de los diferentes componentes, tanto clásicos como cuánticos. Por lo tanto, este pequeño ejemplo de aplicación demuestra la viabilidad y adecuación de la técnica de generación de código para sistemas software híbridos.

5 Conclusiones y trabajo futuro

En este artículo hemos presentado una técnica de generación de código Python (Qiskit) desde modelos UML extendidos para software cuántico. Aunque ya existen multitud de técnicas generativas (siguiendo el enfoque MDE) para software clásico, la contribución principal de esta investigación es que se genera código clásico y cuántico en combinación desde el diseño de alto nivel de sistemas software híbrido. Esta propuesta contribuye así a completar el proceso de modernización de sistemas software híbridos, en los que ya se habían abordado, en mayor o menor medida, las fases de ingeniería inversa y reestructuración.

La transformación de modelos UML a código es preliminar, con lo que todavía existen pequeñas limitaciones, las cuales se proponen como mejoras potenciales para ser abordadas en trabajos futuros. Por ejemplo, contemplar otros elementos en Python y Qiskit, como las librerías de software cuántico predefinidas, así como explorar la generación en otros lenguajes de programación. Se ha de tener en cuenta que no es necesario que el software clásico y cuántico sea generado en el mismo lenguaje de programación. De hecho, la técnica propuesta podría ser modificada para generar el código cuántico orientado a servicios. También, como trabajo futuro, se completará la liberación de la herramienta que integra la transformación EGL a fin de facilitar la realización de varios estudios de casos con modelos UML de sistemas software híbridos.

La mayor implicación de este trabajo es disponer de un enfoque de modernización del software, que sigue los principios MDE, para facilitar la migración de los sistemas existentes. Se considera, por lo tanto, que esto ayudará a que la industria adopte de forma ágil el software cuántico, mientras que se mantiene parte de los sistemas software

clásicos actuales y, sobre todo, se preserva el conocimiento de negocio crítico embebido en el software. Esta adopción acelerará la transición hacia la tecnología/computación cuántica para lograr así las ventajas de muchas de sus aplicaciones.

Agradecimientos. Este trabajo ha sido financiado por los proyectos PID2019-104791RB-I00 (Proyecto SMOQUIN) financiado por MCIN/AEI/10.13039/501100011033; y PDC2022-133051-I00 (Proyecto QU-ASAP) financiado por MCIN/AEI/10.13039/501100011033 y por la “Unión Europea NextGenerationEU/PRTR”.

Referencias

1. M. Allende López and M. M. Da Silva, "Quantum Technologies: Digital Transformation, Social Impact, and Cross-sector Disruption," Inter-American Development Bank, 2019. <https://publications.iadb.org/en/quantum-technologies-digital-transformation-social-impact-and-cross-sector-disruption>
2. IBM, *The Quantum Decade. A playbook for achieving awareness, readiness, and advantage*, 3rd ed. IBM Institute for Business Value, 2022, p. 127.
3. D. Ukpabi *et al.*, "Framework for understanding quantum computing use cases from a multidisciplinary perspective and future research directions," *arXiv*, vol. 2212.13909v2, p. 13, 5 Jan 2023 2023, doi: <https://doi.org/10.48550/arXiv.2212.13909>.
4. D. J. Egger *et al.*, "Quantum Computing for Finance: State-of-the-Art and Future Prospects," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1-24, 2020, doi: 10.1109/TQE.2020.3030314.
5. The Economist. "Wall Street's latest shiny new thing: quantum computing." <https://www.economist.com/finance-and-economics/2020/12/19/wall-streets-latest-shiny-new-thing-quantum-computing> (accedido 07/01/2021, 2021).
6. Y. Cao *et al.*, "Quantum Chemistry in the Age of Quantum Computing," *Chemical Reviews*, vol. 119, no. 19, pp. 10856-10915, 2019/10/09 2019, doi: 10.1021/acs.chemrev.8b00803.
7. Y. Cao, J. Romero, and A. Aspuru-Guzik, "Potential of quantum computing for drug discovery," *IBM Journal of Research and Development*, vol. 62, no. 6, pp. 6:1-6:20, 2018, doi: 10.1147/JRD.2018.2888987.
8. M. Njorbuenwu, B. Swar, and P. Zavarsky, "A Survey on the Impacts of Quantum Computers on Information Security," in *2019 2nd International Conference on Data Intelligence and Security (ICDIS)*, 28-30 June 2019 2019, pp. 212-218, doi: 10.1109/ICDIS.2019.00039.
9. C. Savoie. "How Quantum Computers Could Cut Millions Of Miles From Supply Chains And Transform Logistics." *Forbes*. <https://www.forbes.com/sites/forbestechcouncil/2021/02/05/how-quantum-computers-could-cut-millions-of-miles-from-supply-chains-and-transform-logistics/> (accedido 03/22/2021, 2021).
10. Y. Zhang and Q. Ni, "Recent advances in quantum machine learning," vol. 2, no. 1, p. e34, 2020, doi: <https://doi.org/10.1002/que2.34>.
11. S. Garg and G. Ramakrishnan, "Advances in quantum deep learning: An overview," *arXiv preprint*, vol. arXiv:04316, 2020.
12. B. Lenahan, *Quantum Boost: Using Quantum Computing to Supercharge Your Business*. Library and Archives Canada, 2021.

13. IBM Q, "Introduction to Quantum Circuits. <https://quantum-computing.ibm.com/support/guides/introduction-to-quantum-circuits>," IBM, 2019. <https://quantum-computing.ibm.com/support/guides/introduction-to-quantum-circuits>
14. F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505-510, 2019, doi: 10.1038/s41586-019-1666-5.
15. Livemint. "Google says it will build a commercial quantum computer before 2030." Livemint. <https://www.livemint.com/technology/tech-news/google-says-it-will-build-a-commercial-quantum-computer-before-2030-11621363226405.html> (accedido 2021).
16. M. Piattini *et al.*, "The Talavera Manifesto for Quantum Software Engineering and Programming," presented at the QANSWER 2020. QuANtum SoftWare Engineering & pROgramming, Talavera de la Reina, 2020, 1. <http://ceur-ws.org/Vol-2561/paper0.pdf>.
17. L. Mueck, "Quantum software," *Nature*, vol. 549, no. 7671, pp. 171-171, 2017/09/01 2017, doi: 10.1038/549171a.
18. M. Piattini, G. Peterssen, M. A. Serrano, J. L. Hevia, and R. Pérez-Castillo, "Towards a Quantum Software Engineering," *IT Professional*, vol. 23, no. 1, pp. 62-66, 1 Jan.-Feb. 2021 2021, doi: 10.1109/MITP.2020.3019522.
19. A. D. Carleton *et al.*, "Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research and Development," Software Engineering Institute, Carnegie Mellon University, November 2021 2021. <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=741193>
20. M. A. Serrano, R. Pérez-Castillo, and M. Piattini, Eds. *Quantum Software Engineering*. Springer, 2022, pp. XXVIII, 302. <https://link.springer.com/book/10.1007/978-3-031-05324-5>.
21. J. Zhao, "Quantum Software Engineering: Landscapes and Horizons," *arXiv preprint*, 14 Jul 2020 2020, doi: arXiv:2007.07047v1.
22. R. Pérez-Castillo, M. A. Serrano, and M. Piattini, "Software modernization to embrace quantum technology," *Advances in Engineering Software*, vol. 151, p. 102933, 2021/01/01/ 2021, doi: <https://doi.org/10.1016/j.advengsoft.2020.102933>.
23. R. Pérez-Castillo, L. Jiménez-Navajas, and M. Piattini, "QRev: migrating quantum code towards hybrid information systems," *Software Quality Journal*, 2021/10/29 2021, doi: 10.1007/s11219-021-09574-x.
24. R. Pérez-Castillo, L. Jiménez-Navajas, and M. Piattini, "Dynamic analysis of quantum annealing programs," *Journal of Systems and Software*, vol. 201, p. 111683, 2023/07/01/ 2023, doi: <https://doi.org/10.1016/j.jss.2023.111683>.
25. R. Pérez-Castillo, L. Jiménez-Navajas, and M. Piattini, "Modelling Quantum Circuits with UML," presented at the 43rd ACM/IEEE International Conference on Software Engineering Workshops. 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), Virtual (originally in Madrid, Spain), May 25-28, 2021, 2021, 2.
26. B. Heim *et al.*, "Quantum programming languages," *Nature Reviews Physics*, vol. 2, no. 12, pp. 709-722, 2020/12/01 2020, doi: 10.1038/s42254-020-00245-7.
27. S. Garhwal, M. Ghorani, and A. Ahmad, "Quantum Programming Language: A Systematic Review of Research Topic and Top Cited Languages," *Archives of Computational Methods in Engineering*, vol. 28, no. 2, pp. 289-310, 2021/03/01 2021, doi: 10.1007/s11831-019-09372-6.
28. M. A. Serrano, J. A. Cruz-Lemus, R. Perez-Castillo, and M. Piattini, "Quantum Software Components and Platforms: Overview and Quality Assessment," *ACM Comput. Surv.*, vol. 55, no. 8, p. Article 164, 2022, doi: 10.1145/3548679.



29. A. McCaskey, E. Dumitrescu, D. Liakh, and T. Humble, "Hybrid Programming for Near-Term Quantum Computing Systems," presented at the 2018 IEEE International Conference on Rebooting Computing (ICRC), 7-9 Nov. 2018, 2018.
30. J. A. Cruz-Lemus, L. A. Marcelo, and M. Piattini, "Towards a Set of Metrics for Quantum Circuits Understandability," Cham, 2021.
31. J. Garcia-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, and J. M. Murillo, "Quantum Software as a Service Through a Quantum API Gateway," *IEEE Internet Computing*, vol. 26, no. 1, pp. 34-41, 2022, doi: 10.1109/MIC.2021.3132688.
32. R. Pérez-Castillo and M. Piattini, "Design of classical-quantum systems with UML," *Computing*, 2022/05/31 2022, doi: 10.1007/s00607-022-01091-4.
33. I. Qiskit. "Getting Started with Qiskit." https://qiskit.org/documentation/tutorials/circuits/1_getting_started_with_qiskit.html (accedido 12/12/2022).
34. I. Cantalejo. "EGL scripts for transforming Quantum UML models into Python code." <https://github.com/ivyncm/PythonGenerator/tree/main/EGLtemplates>.
35. N. Dey, M. Ghosh, S. S. kundu, and A. Chakrabarti. "The Quantum Development Life Cycle." <https://arxiv.org/abs/2010.08053v1> (accedido 08/01/2021).
36. *UML 2.5.1*. <https://www.omg.org/spec/UML/2.5.1/PDF>, OMG, 2017. <https://www.omg.org/spec/UML/2.5.1/PDF>
37. C. A. Perez-Delgado and H. G. Perez-Gonzalez, "Towards a Quantum Software Modeling Language," presented at the 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW), Han River, Seoul, South Korea, 2020. <https://arxiv.org/pdf/2006.16690.pdf>.
38. J. M. Ribo and J. Franch, "A two-tiered methodology to extend the UML metamodel." <https://upcommons.upc.edu/bitstream/handle/2117/97437/R02-52.pdf>, Universitat Politècnica de Catalunya, 2002. <https://upcommons.upc.edu/bitstream/handle/2117/97437/R02-52.pdf>
39. R. Pérez-Castillo. "Example of application of quantum UML profile - Pricing European Call Options." <https://github.com/ricpdc/quml-example> (accedido 01/02/2022, 2022).
40. R. G. Alarcos. "Pricing European Call Options App." <http://alarcosj.esi.uclm.es:8080/> (accedido 01/02/2022, 2022).