Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Automatisierte Generierung von Amazon Braket Hybrid Jobs

Sharon-Naemi Stiliadou

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	Martin Beisel, M.Sc., Benjamin Weder, M.Sc.
Beginn am:	2. März 2022
Beendet am:	2. September 2022

Kurzfassung

Heutige Quantencomputer erweitern die Kapazitätsgrenzen klassischer Computer, da Berechnungen basierend auf Qubits erfolgen. Insbesondere kann durch Verschränkung der Qubits die Berechnung auf dem Quantencomputer einen exponentiellen Speedup gegenüber der Berechnung auf dem klassischen Computer erzielen. Jedoch haben heutige Quantencomputer nur begrenzte Möglichkeiten zur Fehlerkorrektur und sind in ihrer Anzahl an Qubits beschränkt. Aus diesem Grund sind weiterhin klassische Computer zur Datenvor- und nachbereitung erforderlich. Quantenalgorithmen, welche aus klassischen und Quanten Anteilen bestehen, werden als hybrid bezeichnet. Diese Quantenalgorithmen können durch Programme beschrieben werden, deren Zugriffsmodell Queue-basiert ist. Dabei werden zur Ausführung dieser Programme Orchestrierungsmechanismen wie Workflows genutzt, die skalierbar und robust sind. Jedoch entstehen gerade bei variationellen Algorithmen große Latenzen, da diese iterativ eine Lösung bestimmen. Somit findet ein wiederholter Datentransfer zwischen klassischen und Quanten Anteilen statt. Diese Problematik lässt sich durch hybride Runtimes lösen, indem klassische Ressourcen nahe an der QPU provisioniert werden und somit die Latenzen minimiert werden. Dabei müssen die hybriden Runtime Programme aus Queue-basierten Programmen automatisiert generiert werden, da die manuelle Transformation zeitintensiv und fehleranfällig ist. Open-Source Projekte wie der Qiskit Runtime Handler erlauben bereits die automatisierte Generierung von hybriden Programmen. Dennoch fehlt derzeit ein Konzept zur Überführung von Amazon Braket Programmen zu Amazon Braket Hybrid Jobs. Im Rahmen dieser Arbeit wird deshalb ein Konzept zur automatisierten Generierung von Amazon Braket Hybrid Jobs vorgestellt, welches prototypisch im Amazon Braket Jobs Handler umgesetzt wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

Amazon S3 Amazon Simple Storage Service. 21

ARN Amazon Resource Name. 35

AWS Amazon Web Services. 15

NISQ Noisy Intermediate-Scale Quantum. 17

QAA Quantum Application Archive. 23

QAOA Quantum Approximate Optimization Algorithm. 18

QCaaS Quantum Computing as a Service. 18

QPU Quantum Processing Unit. 15

QRM QuantME Replacement Model. 22

QuantME Quantum Modeling Extension. 21

VQE Variational Quantum Eigensolver. 15

1 Einleitung

Paradigmen wie Cloud Computing und das Internet der Dinge sind für den Fortschritt in der Industrie und Wissenschaft verantwortlich, da sie die Grundlage für intelligente Systeme sind und Innovationen ermöglichen [TM20]. Gleichzeitig unterliegen diese Paradigmen dem heutigen Computermodell unter Verwendung von Bits, sodass dieses auch der begrenzende Faktor für den Fortschritt ist. Aus diesem Grund gewinnt das Quantencomputing zunehmend an Bedeutung, da es sowohl eine Einsparung in Energie als auch eine erhöhte Genauigkeit verspricht [Bar22; NC10]. Zudem ermöglicht es bestimmte Probleme auf Quantencomputern exponentiell schneller zu lösen als es auf klassischen Computern möglich ist [HTC18]. Dabei sind heutige Quantencomputer in ihrer Anzahl an Qubits beschränkt sowie fehleranfällig. Dies hängt damit zusammen, dass ein reales Quantensystem nicht abgeschlossen ist und mit der Umgebung interagiert, sodass der Zustand des Systems irreversibel verändert und somit gestört wird [Pre18].

Die meisten Quantenalgorithmen bestehen aus Quanten Anteilen und klassischen Anteilen, sodass man diese Quantenalgorithmen als hybrid bezeichnet [WBL+20]. Ein Beispiel ist der *Variational Quantum Eigensolver (VQE)* [KMT+17], indem die Messung der Elemente der Pauli-Gruppe auf der *Quantum Processing Unit (QPU)* und die Summation der Messungen auf der CPU erfolgen [PMS+14]. Dadurch eignen sich hybride Algorithmen für die Ausführung in der Cloud wie *Amazon Web Services (AWS)* durch Amazon Braket. Jedoch ist das Zugriffsmodell Queue-basiert. Dadurch kann es zu einer erhöhten Latenz bei der Ausführung auf der QPU kommen, wenn die Queue voll ist [WBLZ21]. Zusätzlich kann der Datentransfer zwischen klassischen und Quanten Anteilen ineffizient sein. Da die Daten aus der QPU in die Cloud geladen sowie dort verarbeitet werden und wieder zurück zur QPU gesendet werden müssen. Aus diesem Grund wächst die Relevanz von hybriden Runtimes wie der Qiskit Runtime [Qis21] oder Amazon Braket Hybrid Jobs [Poc21]. Da diese die QPU für die Ausführung der Algorithmen reserviert sowie alle angeforderten klassischen Ressourcen hochfährt. Außerdem sind diese Ressourcen möglichst nahe bei der QPU um die Latenz weiter zu verringern. Zudem werden diese wenn sie nicht mehr erforderlich sind, automatisch heruntergefahren, sodass nur Kosten für die Ausführung entstehen. Aufgrund dieser Vorteile ist die Verwendung dieser hybriden Runtimes gegenüber Angeboten der Provider wie Amazon Braket sinnvoll. Jedoch fehlt derzeit ein Konzept zur automatisierten Transformation von Amazon Braket Programmen zu Amazon Braket Hybrid Jobs, da die manuelle Transformation komplex, aufwendig und fehleranfällig ist.

Das Ziel dieser Bachelorarbeit ist es daher, ein Generierungskonzept für die Amazon Braket Hybrid Jobs, welches herkömmliche Amazon Braket Programme in Amazon Braket Hybrid Jobs umwandelt, zu entwickeln. Darauf aufbauend erfolgt die prototypische Implementierung sowie die Integration in das *MODULO Framework* [WBL21]. Dies ermöglicht die Modellierung, Transformation und das Deployment von Quanten Workflows. Zum Schluss erfolgt die Evaluierung des entwickelten Konzepts.

Aufbau der Arbeit

Die Arbeit ist folgendermaßen gegliedert:

Kapitel 2 - Grundlagen:
Dieses Kapitel beinhaltet die grundlegenden Begriffe, welche für das Verständnis der Thematik notwendig sind.

Kapitel 3 - Verwandte Arbeiten:
In diesem Kapitel werden verwandte Arbeiten präsentiert, welche Generierungskonzepte von Workflows thematisieren.

Kapitel 4 - Problemstellung:
Dieses Kapitel stellt die bestehenden Probleme vor, welche durch die Arbeit gelöst werden sollen.

Kapitel 5 - Konzept:
Hier wird das entwickelte Generierungskonzept zur Transformation von Amazon Braket Programmen zu Amazon Braket Hybrid Jobs präsentiert.

Kapitel 6 - Implementierung:
Das Kapitel befasst sich mit der prototypischen Implementierung des in Kapitel 5 vorgestellten Konzepts.

Kapitel 7 - Anwendungsszenario:
In diesem Kapitel erfolgt die Beschreibung eines Szenarios, welches zur Verifikation des Konzepts herangezogen wird.

Kapitel 8 - Evaluation:
Das Kapitel beschäftigt sich mit der Evaluation des entwickelten Konzepts. Dazu wird die Zeit für die Ausführung eines Workflows mit und ohne hybride Runtime verglichen.

Kapitel 9 - Zusammenfassung und Ausblick:
Das letzte Kapitel beinhaltet die Zusammenfassung der wichtigsten Ergebnisse und gibt einen Ausblick auf mögliche Erweiterungen.

2 Grundlagen

In diesem Kapitel werden die Begriffe, die für das Verständnis der Arbeit notwendig sind, erläutert. In den ersten beiden Abschnitten geht es um die Relevanz des Quantencomputings sowie dessen Rolle für die Cloud. Anschließend werden hybride Algorithmen und Herausforderungen bei deren Ausführung vorgestellt. Nachfolgend werden hybride Runtimes betrachtet, die zur Verbesserung der Ausführung von hybriden Quantenalgorithmen führen. Zum Schluss werden Workflows und deren Modellierung durch das MODULO Framework präsentiert, welche die Orchestrierung der Quantenalgorithmen ermöglichen.

2.1 Quantencomputing

Das Paradigma des Quantencomputings gewinnt zunehmend an Bedeutung, da es ermöglicht bestimmte Probleme, wie beispielsweise Suchen [Gro96] oder Faktorisierung [Sho94], auf Quantencomputern schneller zu lösen als es auf klassischen Computern möglich ist [HTC18]. Dies ist auf die grundlegende Informationseinheit des Quantencomputers zurückzuführen, welcher auf Qubits statt Bits basiert. Ein Qubit unterscheidet sich von einem Bit darin, dass ein Bit entweder im Zustand 0 oder 1 sein kann, während ein Qubit sich in der Superposition dieser Zustände befindet. Dabei beschreibt die Superposition die Linearkombination zweier Quantenzustände [JAA+18]. Außerdem können klassische Bits gelesen werden, während ein Qubit gemessen werden muss [Hom18]. Ein klassisches Register mit n Bits kann einen der 2^n Zustände von 0 bis $2^n - 1$ annehmen. Ein Quantenregister mit n Qubits kann sich in 2^n vielen Werten gleichzeitig befinden. Dadurch können überabzählbar viele mögliche Werte simuliert werden, welche gleichzeitig manipuliert werden können. Dies bezeichnet man auch als Quantenparallelismus. Jedoch sind heutige Quantencomputer in ihrer Anzahl an Qubits und in ihrer Beziehung zueinander beschränkt [HTC18]. Dies führt dazu, dass Quantencomputer nicht beliebig skalierbar sind. Zudem ist ein reales Quantensystem nicht abgeschlossen und interagiert mit der Umgebung, wodurch Qubits sowie Operationen der QPU fehleranfällig sind [PMS+14]. Dadurch sind Qubits nur für eine bestimmte Zeit stabil bevor sie zerfallen. Dieser Prozess wird als Dekohärenz bezeichnet [NC10]. Diese Problematik wird auch von Preskill [Pre18] identifiziert, welcher den Begriff der *Noisy Intermediate-Scale Quantum (NISQ)* prägte unter dem man Quantenrechner mit 50–100 Qubits beschreibt, die nur über begrenzte Möglichkeiten zur Fehlerkorrektur verfügen. Die Fehler können dabei während der Ausführung sowie mit klassischen Nachbearbeitungsschritten mitigiert werden.

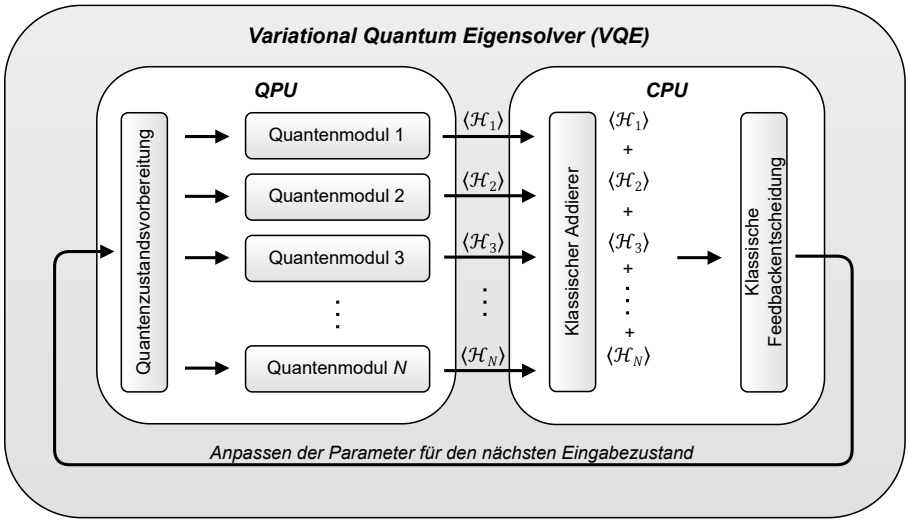

Abbildung 2.1: Ablauf der Berechnung des VQEs [PMS+14]

2.2 Quantencomputing in der Cloud

Cloud Computing und Quantencomputing sind Paradigmen, welche die Industrie und Wirtschaft revolutionieren [TM20]. Allgemein ermöglicht Cloud Computing die elastische Bereitstellung von Rechenleistung und Speicherplatz. Hierbei werden IT-Ressourcen automatisch nach Bedarf provisioniert, wodurch Über-und Unterprovisionierung verhindert werden. Hingegen kann beim Quantencomputing eine große Menge an Daten in einem Schritt verarbeitet und komplexe Probleme effizient gelöst werden [Vog11]. Jedoch sind Quantencomputer für Unternehmen schwer verfügbar und teuer [SS14]. Aus diesem Grund haben sich Cloud Angebote wie *Quantum Computing as a Service (QCaaS)* etabliert, welche bereits durch Cloud Anbieter wie IBM oder AWS realisiert werden. Bei QCaaS handelt es sich um ein Servicemodell, in dem ein Simulator oder ein Quantencomputer in der Cloud zur Verfügung gestellt wird [RI16]. Somit können Nutzer vom bedarfsorientierten Preismodell profitieren als auch von einem Speedup für bestimmte Probleme. Die Nachteile, die dafür in Kauf genommen werden müssen, sind die Abhängigkeit vom Provider sowie deren zugehörigen Hersteller der Quantencomputer [Ley21].

2.3 Hybride Algorithmen

Heutige Quantencomputer sind in ihrer Anzahl an Qubits beschränkt sowie fehleranfällig [Pre18]. Das Mitigieren dieser ist komplex und aufwendig, deshalb empfiehlt es sich die Zeit auf dem NISQ-Quantencomputer zu reduzieren [WBLV21b]. Das bedeutet, die Ausführung soll auf klassischen Ressourcen sowie auf Quantencomputern stattfinden. Diese Quantenalgorithmen werden als hybrid bezeichnet. So werden insbesondere die Vorverarbeitung und Nachbearbeitung auf klassischen Computern ausgeführt, während die Zustandsvorbereitung sowie die unitäre Transformation und Messung auf dem Quantencomputer erfolgt [WBLV21a]. Beispiele für hybride Algortihmen sind VQE und *Quantum Approximate Optimization Algorithm (QAOA)* [FGG14], welche Optimierungsalgorithmen darstellen. Der schematische Ablauf des VQE Algorithmus ist in Abbildung 2.1 zu

sehen. Hierbei wird die Trennung zwischen den klassischen und Quanten Anteilen deutlich. Dabei erfolgt die Zustandsvorbereitung auf dem Quantencomputer, das heißt es wird ein parametrisierter Quantenschaltkreis erzeugt, welcher den zu minimierenden Ansatz realisiert. Im nächsten Schritt werden die Erwartungswerte der verschiedenen Quantenmodule, welche den Pauli-Matrizen entsprechen, berechnet. Diese Erwartungswerte werden auf der CPU summiert und basierend auf deren Ergebnis bestimmt eine klassische Optimierungsmethode die Wahl der Parameter für die nächste Iteration bis das Verfahren konvergiert. Somit werden nur die Teile auf dem Quantencomputer ausgeführt, welche einen exponentiellen Speedup erzeugen. Jedoch erfordert diese Auftrennung in klassische und Quanten Anteile Verständnis in Mathematik, Informatik sowie Physik und ist bei manueller Durchführung fehleranfällig und zeitintensiv [NC10]. Ein Lösungsansatz ist das Verwenden von Patterns, welche Richtlinien beschreiben, wie die Trennung von klassischen und Quanten Anteilen erfolgen kann [WBLV21b].

2.4 Hybride Runtimes

Die Ausführungszeit eines hybriden Quantenalgorithmus wird durch verschiedene Latenzen beeinflusst. Zum einen entstehen Latenzen, wenn der vom Nutzer erstellte Quantenschaltkreis an die Cloud gesendet wird, da diese häufig geographisch entfernt ist. Zum anderen kann der Auftrag für einen Quantencomputer erst erfolgen, wenn er sich an der aktuellen Verarbeitungsposition der Queue der jeweiligen QPU befindet [KTP+20]. Im Falle einer leeren Queue, wird der Auftrag direkt ausgeführt. Jedoch müssen diese Ergebnisse wieder an die Cloud zurückgesendet werden, welches

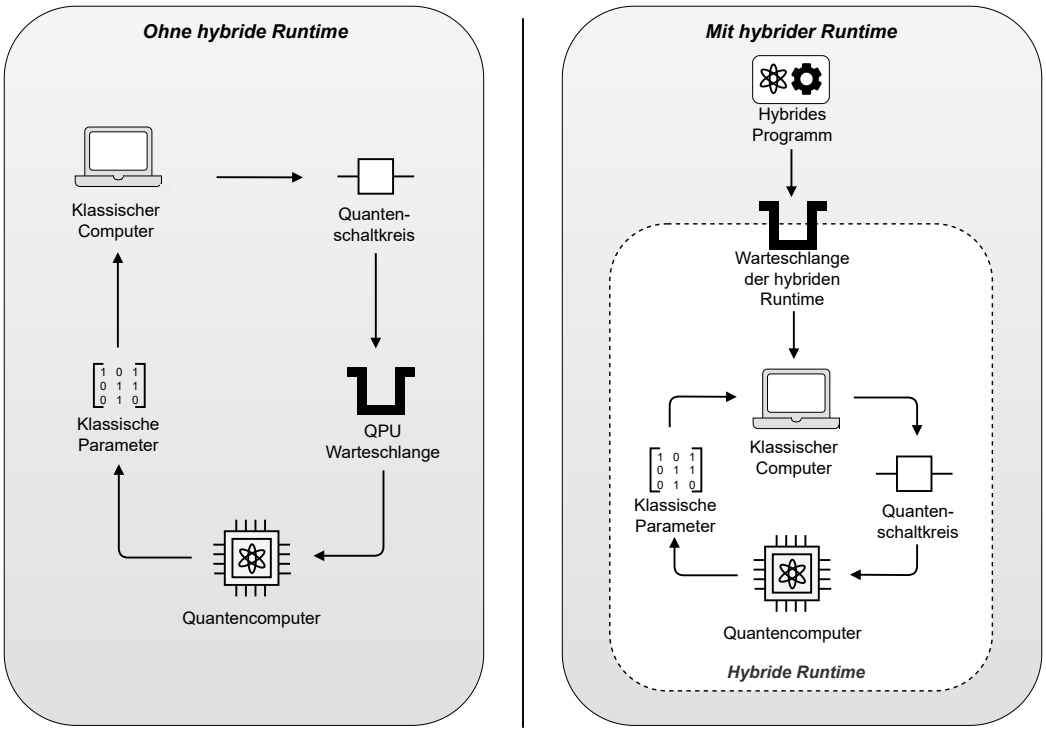

Abbildung 2.2: Ablauf eines hybriden Quantenprogramms ohne hybride Runtime (links) und mit hybrider Runtime (rechts)

ebenfalls Latenzen erzeugt. Insbesondere sind bei *variationellen Algorithmen* [CAB+21] viele Iterationen erforderlich, in denen die Parameteroptimierung im Vordergrund steht. Das bedeutet, es entstehen pro Iterationen die Latenzen zur Cloud und zurück sowie auch Verzögerungen durch das Warten in der Queue, da diese keinen Prioritätszugriff auf die jeweilige QPU hat. Aus diesem Grund wächst die Relevanz von hybriden Runtimes wie Qiskit Runtime[1] oder Amazon Braket Hybrid Jobs. Diese zeichnen sich dadurch aus, dass die QPU für die Ausführung der Algorithmen reserviert wird und die klassischen Ressourcen möglichst nahe an die QPU gebracht werden, um die Latenzen zu verkürzen.

2.4.1 Amazon Braket Hybrid Jobs

Amazon Braket ist ein verwalteter Quanten Service von AWS, der seinen Namen von der Bra-Ket Notation erhielt [Dir39]. Dazu kann der Nutzer ein Amazon Braket Programm entweder direkt über Jupyter Notebook verwendet werden oder durch die Amazon-Braket-SDK[2] gestartet werden. Dabei handelt es sich um eine Open-Source Python Bibliothek, welche es ermöglicht Quantenschaltkreise lokal zu definieren, die im nächsten Schritt zum Quantencomputer gesendet werden. Außerdem werden Hardware als auch Software Werkzeuge von Rigetti[3], IonQ[4], OQC[5] and D-Wave[6] in der Cloud zur Verfügung gestellt. Diese Ressourcen werden dem Nutzer mittels eines Pay-Per-Use Modells angeboten. Das bedeutet, es werden lediglich die Ressourcen bezahlt, die letztendlich

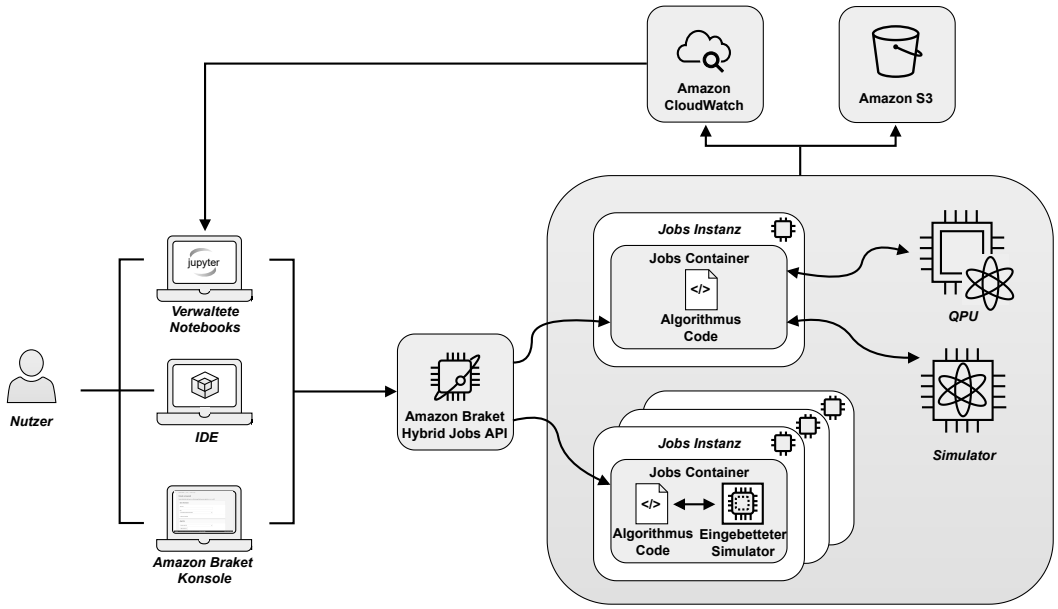

Abbildung 2.3: Überblick über die Amazon Braket Hybrid Jobs Architektur [Ama20]

[1] https://quantum-computing.ibm.com/lab/docs/iql/runtime
[2] https://github.com/aws/amazon-braket-sdk-python
[3] https://qcs.rigetti.com/qpus
[4] https://ionq.com/best-practices
[5] https://oxfordquantumcircuits.com/oqc-on-aws
[6] https://support.dwavesys.com/hc/en-us/articles/360005268633-QPU-Specific-Physical-Properties

genutzt wurden. Jedoch ist das Verwenden von Amazon Braket für den Einsatz von hybriden Algorithmen nicht geeignet beziehungsweise ineffizient. Dies liegt daran, dass im Falle eines variationellen Algorithmus der jeweilige Amazon Braket Auftrag bei nicht leerer Queue warten muss, da kein Prioritätszugang zur QPU bereitstellt wird. Aus diesem Grund wurden im November 2021 die Amazon Braket Hybrid Jobs eingeführt. Die Grundfunktionen eines Amazon Braket Hybrid Jobs sind der Prioritätszugriff auf die jeweilige QPU und die automatische Provisionierung der Ressourcen, wodurch Kosten minimiert werden. In Abbildung 2.3 ist der Ablauf eines Amazon Braket Hybrid Jobs zu sehen. Dazu wird im ersten Schritt ein Quantenalgorithmus durch den Benutzer definiert. Dies kann entweder durch die Amazon-Braket-SDK erfolgen oder durch ein benutzerdefiniertes Docker-Container-Image. Anschließend wird über die Amazon-Braket-API definiert, welches Zielquantengerät (Simulator oder QPU) verwendet werden soll. Zudem erfolgt die Auswahl der klassischer Ressourcen oder der Datenspeicherorte. Nachdem ein Simulator ausgewählt wurde, beginnt sofort die Ausführung. Wenn der Amazon Braket Hybrid Jobs auf einer QPU ausgeführt werden soll, wird dieser erst ausgeführt, wenn die QPU verfügbar ist und der Job an erster Stelle in der Warteschlange liegt. Anschließend werden die Ergebnisse in *Amazon Simple Storage Service (Amazon S3)*[7] zur Verfügung gestellt und die Logging-Einträge sind in Amazon Cloudwatch[8] sichtbar.

2.5 Workflow

Quantenalgorithmen sind häufig hybrid, da sie aus klassischen Vor- und Nachbereitungsschritten sowie Berechnungen auf dem Quantencomputer bestehen. Dies erfordert eine effiziente Ausführung und Orchestrierung zwischen den klassischen und Quanten Anteilen, da sich diese in unterschiedlichen Umgebungen befinden. Dazu eignen sich Workflows, welche robust, verlässlich und skalierbar sind [LR00]. Ein Workflow besteht aus Aktivitäten, die durch Kontrollflusskanten verbunden sind. Zusätzlich kann der Austausch von Daten zwischen Aktivitäten durch Datenflusskanten modelliert werden. Jedoch verfügen standardisierte Workflowsprachen wie BPEL [OAS07] und BPMN [OMG11] derzeit nicht über Modellierungskonstrukte für Quantenberechnungen.

Aus diesem Grund wurde *Quantum Modeling Extension (QuantME)* eingeführt, welches die imperativen Workflowsprachen um die Modellierung von Quantenberechnungen erweitert [WBLW20]. Dazu werden insgesamt sechs Tasks und zwei Datenobjekte als Modellierungskonstrukte in BPMN hinzugefügt, welche in Abbildung 2.4 zu sehen sind. Hierbei kann jede Task einer Phase des Quanten Software Lifecycles zugeordnet werden. Die Datenobjekte stellen die Eingabe und Ausgabe sowie die benötigten Daten zum Ausführen der Tasks dar. Beispielsweise visualisiert die *QuantumCicuit-ExecutionTask* die Ausführung eines Quanten Algorithmus, welcher durch einen Quantenschaltkreis implementiert wurde. Diese Task erhält als Eingabedaten das Datenobjekt *QuantumCircuitObject* und liefert als Ausgabe das Datenobjekt *ResultObject*. Damit die Portabilität des Workflows weiterhin erhalten bleibt, muss das QuantME Workflow Modell zu einem nativen Workflow transformiert werden [WBLS21]. Das native Modell enthält nur die Modellierungskonstrukte der standardisierten imperativen Workflowsprache und kann somit auf der Workflow Engine ausgeführt werden. Zudem

[7] https://aws.amazon.com/de/s3
[8] https://aws.amazon.com/de/cloudwatch

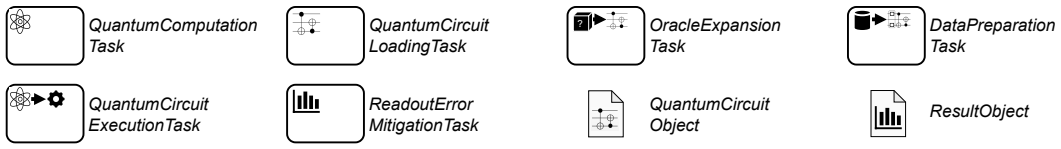

Abbildung 2.4: QuantME Modellierungskonstrukte [WBLW20]

müssen vor der Ausführung des Workflows das Deployment und Binding der Services erfolgen, womit sich das MODULO Framework beschäftigt. Die Details zum MODULO Framework werden im nächsten Abschnitt erläutert.

2.6 MODULO Framework

Damit Quantenalgorithmen von den Vorteilen von Workflows, wie zum Beispiel Skalierbarkeit und Robustheit, profitieren können, können Modellierungserweiterungen wie QuantME genutzt werden [LR00; WBLW20]. Jedoch müssen zusätzlich Services aufgerufen werden, die beispielsweise vor der Ausführung des Workflows manuell deployt und gebunden werden müssen. Das manuelle Binding ist zeitintensiv und fehleranfällig [NSL+14]. Diese Aufgaben können durch das MODULO Framework automatisiert werden. Das MODULO Framework ist eine Erweiterung des Camunda Workflowsystems, welches aus einem BPMN Modeler, einer BPMN Workflowengine und Erweiterungen des OpenTOSCA Ökosystems besteht [WBL21]. Damit die Modellierung, Transformation und das Deployment von Quanten Workflows möglich sind, werden fünf Schritte benötigt, welche in Abbildung 2.5 dargestellt sind:

1) *Modeling:* In diesem Schritt wird der Quanten Workflow modelliert.
2) *Tranformation:* QuantME Aktivitäten werden iterativ durch Workflowfragmente ersetzt, die auch als *QuantME Replacement Models (QRMs)* bezeichnet werden. Der Ersetzungskandidat kann wieder ein QuantME Modellierungskonstrukt oder ein BPMN Task sein. Somit erfolgt die Ersetzung bis ein BPMN natives Workflowmodell erreicht wird.

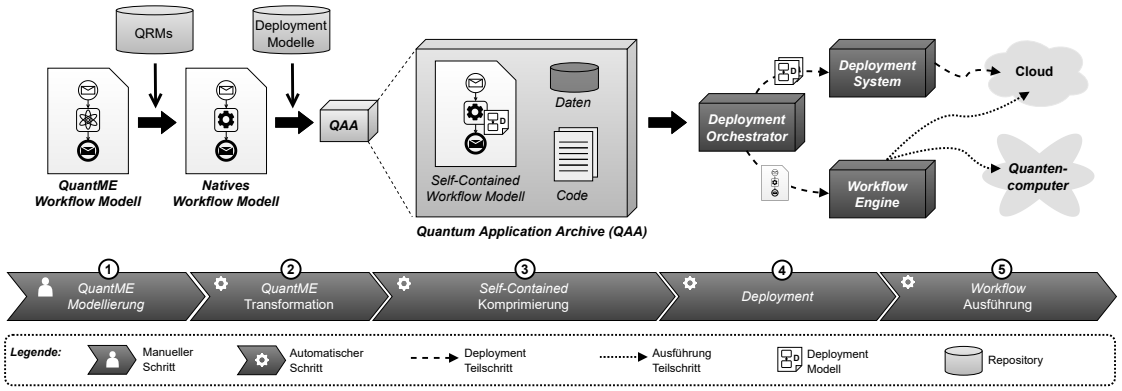

Abbildung 2.5: Modellierung, Transformation und Deployment von Quanten Workflows unter der Verwendung des MODULO Frameworks [WBL21]

3) *Self-Contained Packaging:* Die für das Deployment benötigten Informationen wie klassische und Quanten Programme, Daten, Topologie- und Workflowmodell werden in ein *Quantum Application Archive (QAA)* gepackt [WBLZ21].

4) *Deployment:* Das Deployment der einzelnen Services erfolgt durch einen Deployment Orchestrator, welcher die Deployment Modelle aus den QAA extrahiert und an das jeweilige Deploymentsystem weitergibt.

5) *Workflow Execution:* Der Kontrollfluss und die klassischen Anteile werden beispielsweise auf der Cloud ausgeführt und die Quantenprogramme auf dem Quantencomputer.

3 Verwandte Arbeiten

Dieses Kapitel stellt verwandte Arbeiten aus den Bereichen der Workflow Generierung und Transformation vor. Dabei werden Ansätze vorgestellt, welche die Veränderung von Workflows durch Ersetzen oder Umschreiben von Workflowfragmenten umfassen. So können gesamte Workflows oder zuvor generierte Workflowfragmente in einen existierenden Prozess miteingebunden werden. Abschließend wird die Generierung und Veränderung von Quanten Workflows vorgestellt.

Sun et al. [SKY06] stellen in ihrer Arbeit das Konzept zur Vereinigung von Workflows vor. Dabei bezeichnet man als Workflow Vereinigung den Prozess, welcher durch Hinzufügen von Kanten ein Workflow Schema in ein anderes kombiniert. Dazu wird jede Vereinigung, welche aus zwei Workflows sowie zwei Merge Points besteht, in zwei Dimensionen eingeordnet. In der ersten Dimension wird zwischen verlustfrei und nicht verlustfreien Vereinigungen unterschieden. Ein verlustfreies Zusammenfügen beschreibt die Verbindung von zwei Workflows, deren Ergebnis alle Tasks der einzelnen Workflows enthält. Hingegen wird bei einem nichtverlustfreien Zusammenfügen nicht garantiert, dass alle Tasks im Ergebnis vorhanden sind. Die zweite Dimension enthält vier Merge Patterns, welche klassische Prozesspatterns spezifizieren: sequentiell, parallelel, konditionell und iterativ. Insbesondere wird für komplexere Workflows die Kombination der Merge Patterns erforderlich. Eine Einschränkung bei diesem Ansatz ist, dass nicht untersucht wird, wie sich das Verhalten durch das Einfügen der neuen Kanten zwischen den Merge Points verändert.

Diese Herausforderung wird von La Rosa et al. [LDUD10] gelöst, indem Herkunftsdaten (*Provenance Data*) auf jeder Kante hinzugefügt werden, sodass auf die eingegebenen Workflows zurückgeführt werden kann. Jedoch erzeugt dies eine große Menge an Daten im Workflow, obwohl nur eine Teilmenge an Kanten erforderlich ist, um die Eingabeworkflows zu rekonstruieren. Aus diesem Grund präsentieren Zemni et al. [ZHM14] einen vereinfachten Ansatz, welcher Fragment Adjazenzmatrizen einführt. Eine Fragment Adjazenzmatrix ist eine quadratische Matrix, deren Einträge eine Verbindung von Aktivitäten durch einen Kontrollfluss im Workflow darstellt. Jedoch sind diese Ansätze darin beschränkt, dass Veränderungen nur bei der Erstellung des Workflows möglich sind. Hingegen werden in den Arbeiten von Mundbrod et al. [MGKR15] und Képes et al. [KBG+16] Ansätze vorgestellt, welche den Workflow zur Laufzeit verändern, indem Workflowfragmente kontext- bzw. situtatiosabhängig injeziert werden.

Während die vorgestellten Methoden den Workflow dynamisch durch existierende Workflowfragmente verfeinern, führen Weder et al. [WBBL22] durch automatisierte Programmgenerierung eine Abstrahierung ein. Das Ziel des Konzepts, welches in Abbildung 3.1 dargestellt ist, ist die Verbesserung der Ausführung von hybriden Quantenalgorithmen durch hybride Runtimes. Dies erfolgt durch ein sechs Schritte Phasenmodell. In der ersten Phase wird ein Workflow modelliert, welcher mindestens einen Quanten Task und eine Service Task enthält. Aus diesem Workflow können im zweiten Schritt Ersetzungskandidaten bestimmt werden, wenn Schleifen oder verschachtelte Ausführungen zwischen Quanten Anteilen und klassischen Anteilen existieren. Anschließend erfolgt die Filterung der Kandidaten basierend auf den Eigenschaften der hybriden Runtime und der

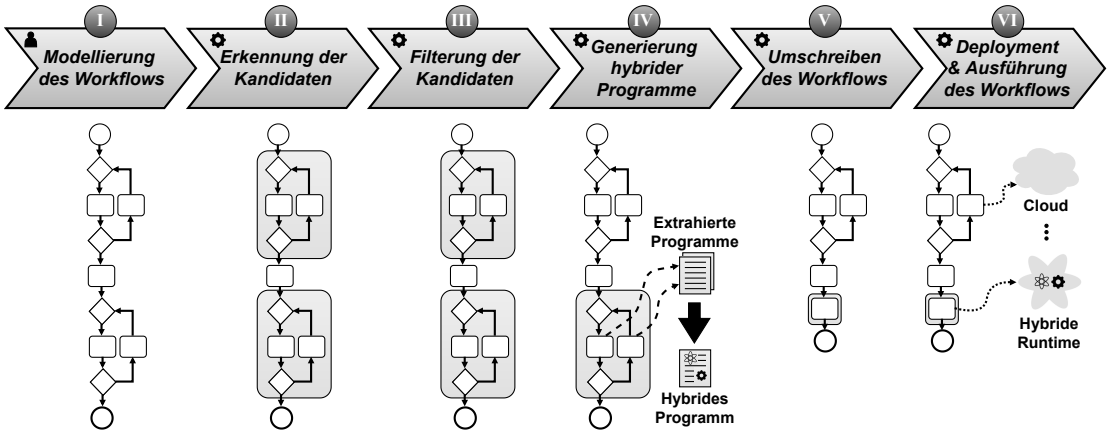

Abbildung 3.1: Analyse und Umschreiben von Quanten Workflows [WBBL22]

unterstützten hybriden Programmen. Die Filterung wird folgendermaßen analysiert: Die Nutzung verschiedener Programmiersprachen ist beschränkt. Durch das Verwenden einer hybriden Runtime ist das Verwenden von Gateways, Events und Aktivitätstypen nur beschränkt möglich, da eine Abhängigkeit zu dem jeweiligen Provider der hybriden Runtime besteht. Im nächsten Schritt wird entsprechend der gewählten hybriden Runtime ein hybrides Programm generiert. Dazu müssen sowohl die Eingabe- und Ausgabeparameter neu bestimmt werden, da die einzelnen Programme zu einem hybriden Programm kombiniert werden. Abschließend wird der Workflow deployt. Dabei werden automatisiert klassische Teile beispielsweise in der Cloud hochgeladen sowie hybride Programme von einer hybriden Runtime ausgeführt. Für die Ausführung in der hybriden Runtime wird in dieser Zeit die QPU reserviert und die klassischen Ressourcen nach Bedarf adaptiert.

4 Problemstellung und Anforderungen

In diesem Kapitel wird die Problemstellung der Arbeit sowie die zu erfüllenden Anforderungen vorgestellt. Hierbei werden Zusammenhänge von Quanten Programmen und deren Ausführung mithilfe von Workflows beschrieben sowie die daraus entstehenden Herausforderungen erläutert. Nachfolgend werden Anforderungen abgeleitet, welche die Kriterien der Arbeit zusammenfassen.

4.1 Problemstellung

In diesem Abschnitt wird die Problemstellung dieser Arbeit sowie die Notwendigkeit zur automatisierten Generierung von hybriden Programmen erläutert. Hierbei besteht die derzeitige Problematik darin, dass aktuell nur die manuelle Transformation von Amazon Braket Programmen zu Amazon Braket Hybrid Jobs möglich ist.

Ein Quantenalgorithmus ist häufig hybrid, sodass die Orchestrierung zwischen den klassischen und Quanten Anteilen durch Workflows erfolgen kann (siehe Abschnitt 2.5). Ein beispielhafter Quanten Workflow ist in Abbildung 4.1 dargestellt. Hierbei führt eine Nachricht mit den Eingabedaten zur Instanziierung des Workflows [WBBL22]. Nachfolgend wird der Algorithmus vorbereitet, indem Parameter und Quantenschaltkreise initialisiert werden. Danach erfolgen Schritte, die für einen variationellen Quanten Algorithmus typisch sind. Dabei werden iterativ Quanten und klassische Teile ausgeführt, indem Parameter dem Quantenschaltkreis übergeben werden und das Ergebnis der Berechnung anhand einer Kostenfunktion bewertet wird. Diese Schritte werden erneut ausgeführt sowie die Parameter adaptiert, wenn die Kosten der Berechnung nicht gegen einen Schwellenwert konvergieren. Sonst terminiert der Algorithmus und sendet dem Nutzer die Ausgabeparameter. Jedoch werden die klassischen als auch die Quanten Anteile in unterschiedlichen Umgebungen ausgeführt, sodass hohe Latenzen entstehen können.

Aus diesem Grund sind Schleifen, in denen sich Quanten und klassische Anteile abwechseln, Ersetzungskandidaten für hybride Programme. Dabei werden für die Ausführung der hybriden Programme hybride Runtimes eingesetzt, welche eine effizientere Ausführung ermöglichen. Dies liegt daran, dass für die Ausführung des Quanten Anteils die QPU reserviert wird sowie die klassischen Ressourcen nach Bedarf provisioniert werden. Um ein hybrides Programm zu erzeugen, müssen die Implementierungen der einzelnen Aktivitäten zusammengefügt werden. Jedoch erfordert das Umschreiben das Verständnis der einzelnen Programme. Insbesondere müssen Eingabe- und Ausgabeparameter adaptiert werden. Außerdem muss der gesamte Prozess wiederholt werden, wenn Aktivitäten hinzugefügt oder gelöscht werden. Aus diesem Grund ist die manuelle Transformation eines Quanten Programms in ein hybrides Programm zeitintensiv und fehleranfällig [NSL+14]. Aufgrund dessen ist ein Konzept zur automatisierten Generierung dieser hybriden Programme erforderlich.

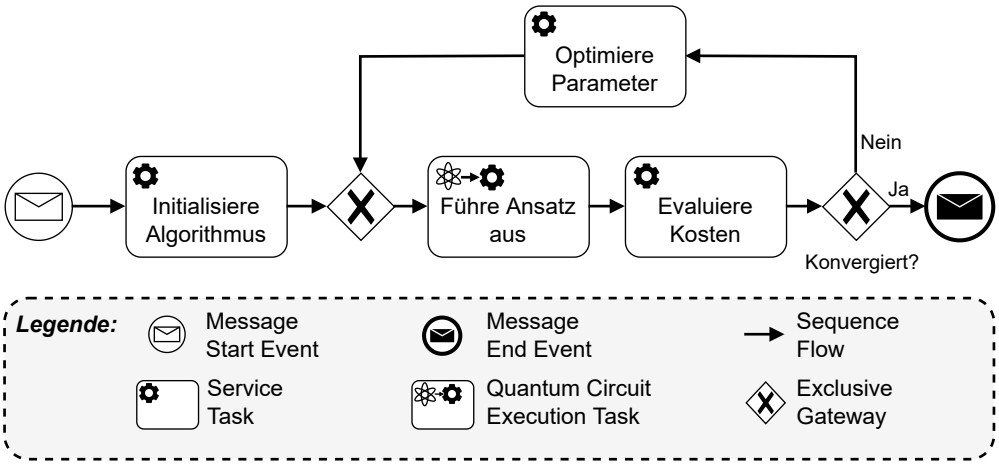

Abbildung 4.1: Quanten Workflow für einen variationellen Algorithmus ([VBL+21; WBBL22])

4.2 Anforderungen

In diesem Abschnitt werden Anforderungen an diese Arbeit definiert. Dabei werden Anforderungen an das Gesamtkonzept zur automatisierten Generierung von Amazon Braket Hybrid Jobs präsentiert. Zudem werden Anforderungen an die Implementierung dieses Konzepts vorgestellt.

- **Anforderung 1 (A1):** Das Konzept soll die automatische Generierung von Amazon Braket Hybrid Jobs basierend auf Amazon Braket Programmen ermöglichen.
- **Anforderung 2 (A2):** Entwicklung einer prototypischen Implementierung des Konzepts.
- **Anforderung 3 (A3):** Integration des Konzepts und der Implementierung mit dem MODULO Transformation Framework.
- **Anforderung 4 (A4):** Evaluation der Performanz von Quanten Workflows, welche aus Amazon Braket Programmen bestehen.

Anforderung **A1** gibt der Arbeit den grundsätzlichen Rahmen. Dabei sollen Amazon Braket Programme so zusammengefasst werden, dass sie ein valider Amazon Braket Hybrid Jobs sind. Ein valides Hybrid Job Programm muss aus einem Startpunkt, den notwendigen imports sowie Eingabe- und Ausgabeparameter bestehen. Die Anforderungen **A2**, **A3** und **A4** beziehen sich auf die prototypische Implementierung des entwickelten Konzepts. Hierbei soll ein Handler erstellt werden, welcher das Konzept umsetzt **(A2)**. Die Implementierung soll dabei auf den in Kapitel 3 vorgestellten Qiskit Runtime Handler[1] basieren, welcher hybride Qiskit Programme generiert. Außerdem soll eine Integration in das MODULO Framework erfolgen, sodass sowohl das Umschreiben von Optimierungskandidaten möglich ist als auch das automatisierte Ausführen von Amazon Braket Hybrid Jobs **(A3)**. Diese Funktionalität soll basierend auf einer Evaluation der Performanz von Quanten Workflows ausgewertet werden **(A4)**.

[1] https://github.com/UST-QuAntiL/qiskit-runtime-handler

5 Konzept

Dieses Kapitel präsentiert ein Konzept zur automatisierten Generierung von Amazon Braket Hybrid Jobs basierend auf Amazon Braket Programmen. Dazu wird zunächst das Generierungskonzept für diese hybriden Programme beschrieben. Abschließend wird das Konzept in die bestehende Architektur eingeordnet und erläutert, welche Vorteile sich daraus ergeben.

5.1 Generierungskonzept

Die automatisierte Generierung von Amazon Braket Hybrid Jobs aus Amazon Braket Programmen erfolgt in drei Schritten, die in Abbildung 5.1 zu sehen sind. Die Eingabe erhält der Amazon Braket Hybrid Jobs Handler[1] von dem *QuantME Transformation Framework*. Dabei besteht die Eingabe aus den Aktivitäten innerhalb der Schleife, sowie der Schleifenbedingung und den benötigten Programmen. Die Aktivitäten innerhalb der Schleife werden in *beforeLoop* und *afterLoop* Aktivitäten unterteilt. BeforeLoop Aktivitäten sind Aktivitäten, die sich zwischen Schleifenanfang und Schleifenbedingung befinden. Hingegen werden die Aktivitäten, welche von der Schleifenbedingung zum Schleifenanfang zurückführen, als afterLoop Aktivitäten bezeichnet. Die benötigten Programme sind die jeweiligen Implementierungen der Aktivitäten, welche in ein hybrides Programm zusammengefügt werden.

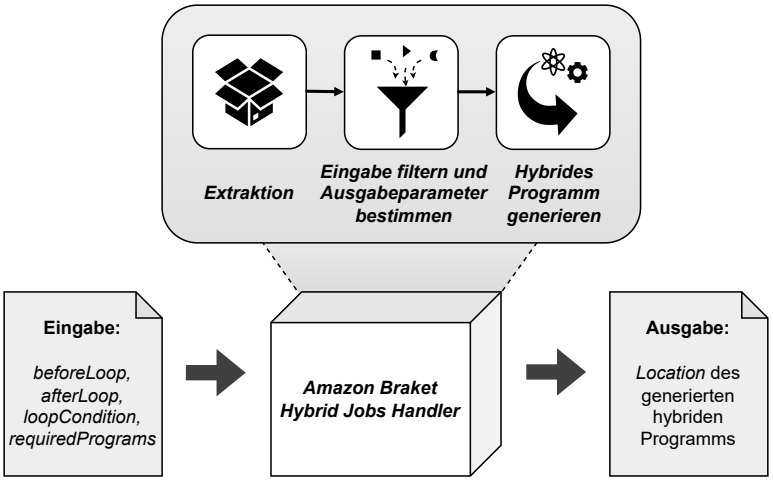

Abbildung 5.1: Generierungskonzept

[1] https://github.com/UST-QuAntiL/amazon-braket-hybrid-jobs-handler

Anschließend erfolgt die Generierung des hybriden Programms im Handler. Diese gliedert sich in drei Schritte. Im ersten Schritt wird für jede Aktivität in der beforeLoop und afterLoop das Programm extrahiert. Danach werden die Eingabe- und Ausgabeparameter bestimmt. Dabei entfallen die Eingabeparameter, welche Ausgabeparameter der vorherigen Aktivität sind. Dies gilt analog für die Ausgabeparameter. Im letzten Schritt werden die Programme basierend auf ihrer Reihenfolge im Workflow in ein hybrides Programm kombiniert. Dazu wird zunächst das Template für den hybriden Job eingelesen und schrittweise ergänzt. Dabei werden für die Generierung Templates verwendet, da Amazon Braket Hybrid Jobs immer aus den Abhängigkeiten, der Startmethode und der Programmlogik aufgebaut sind. Somit unterstützen die Templates die Wiederholbarkeit und Effizienz der Generierung. Anschließend erfolgt die Generierung des hybriden Programms beginnend mit den beforeLoop Aktivitäten, gefolgt von den afterLoop Aktivitäten. Dazu wird für jede Aktivität die Abhängigkeit zum Template hinzugefügt. Im nächsten Schritt wird überprüft, ob eine execute Methode in der Implementierung der Aktivität zu finden ist, da diese den Startpunkt markiert. Im Anschluss daran werden die Methoden rekursiv hinzugefügt. Dazu wird für jede Methode überprüft, ob diese bereits zum Template hinzugefügt wurde. Außerdem wird der Methodennamen um die Aktivitäts ID ergänzt, um Namenskollisionen zu vermeiden. Daraufhin wird die main Methode um die Eingabeparameter ergänzt. Anschließend wird in der Programmschleife die Reihenfolge des Workflows realisiert. Zum Schluss wird die Methode um das Speichern der Ausgabeparameter erweitert. Somit wurde ein valides hybrides Programm erstellt, welches zu einer Zipdatei komprimiert werden kann. Anschließend wird für die spätere Kommunikation mit der Camunda Engine das polling-agent Template verwendet. Dieses enthält das Hochladen des hybriden Programms in Amazon S3, das Senden des hybriden Programms zur Ausführung als auch die Behandlung der Ausgabeparameter. Ist dieses erfolgreich generiert, wird es zu einer Zipdatei komprimiert und das Dockerfile wird ergänzt. Das Dockerfile enthält die erforderlichen Installierungsschritte, damit das hybride Programm später in der Camunda Engine ausgeführt werden kann. Zum Schluss sendet der Handler die URL zurück, an dem sich das hybride Programm befindet. Diese URL empfängt das QuantME Transformation Framework und tauscht den Ersetzungskandidat durch eine einzelne Task aus.

5.2 Architektur

In Abbildung 5.2 ist die Erweiterung des MODULO Frameworks zu sehen. Allgemein ist das Framework für die Modellierung, Transformation und das Deployment von Quanten Workflows zuständig [WBL21]. Zudem können Workflowfragmente, wie in Kapitel 3 beschrieben, analysiert und ersetzt werden. Dabei wurde das QuantME Transformation Framework um die Kommunikation mit dem Amazon Braket Hybrid Jobs Handler erweitert. Dazu wurde der `AWSRuntimeHandler` erstellt, welcher das Umschreiben von Ersetzungskandidaten im Workflows basierend auf Amazon Braket Hybrid Jobs ermöglicht. Zusätzlich wurde auch das Konfigurieren des jeweiligen *Endpoints* für den Amazon Braket Hybrid Jobs Handler ergänzt. Der Handler besteht aus drei wesentlichen Komponenten [WBBL22]:

- *Input Parser:* Basierend auf den Task Namen im Workflow werden die Amazon Braket Programme extrahiert.

- *Hybrid Program Generator:* In diesem Schritt werden Eingabe- und Ausgabeparameter bestimmt. Außerdem findet die Generierung des hybriden Programms basierend auf dem Kontrollfluss statt.
- *Templates:* Enthält die Vorlagen für das hybride Programm, das Dockerfile und den `polling-agent` für den Austausch mit dem MODULO Framework.

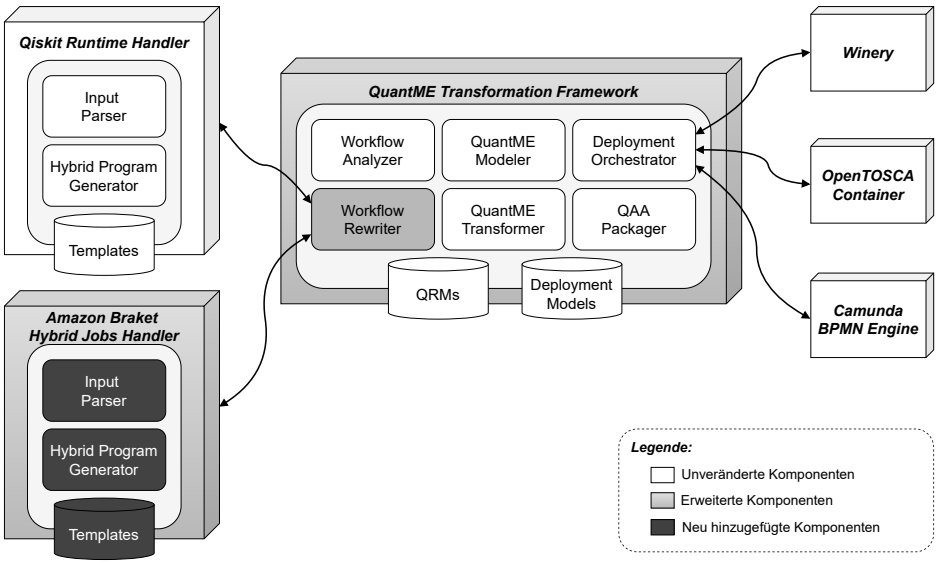

Abbildung 5.2: Erweiterung des MODULO Frameworks um den Amazon Braket Hybrid Jobs Handler (adaptiert von [WBBL22])

6 Implementierung

Dieses Kapitel beinhaltet die prototypische Implementierung des in Kapitel 5 beschriebenen Konzepts. Dabei werden die Implementierung des MODULO Frameworks und des Qiskit Runtime Handlers als Basis verwendet. Hierbei soll die Implementierung des Qiskit Runtime Handler so angepasst werden, dass sich eine neue Komponente ableiten lässt, welche aus Amazon Braket Programmen Amazon Braket Hybrid Jobs erstellt. Zunächst werden die Anforderungen an die Amazon Braket Programme vorgestellt. Anschließend wird der Amazon Braket Hybrid Jobs Handler beschrieben. Damit das Deployment sowie die Ausführung des Quanten Workflows stattfinden kann, muss die Kommunikation zwischen dem Amazon Braket Hybrid Jobs Handler und dem QuantME Transformation Frameworks sichergestellt werden. Zusätzliche Details der Implementierungen werden in den folgenden Abschnitten vorgestellt.

6.1 Annahmen über Amazon Braket Programme

Die Nutzung von AWS Ressourcen ist nach Anlegen eines AWS Accounts mit gültiger Kreditkarte möglich. Durch Anlegen des Accounts können Schlüssel für die Kommunikation mit AWS generiert werden. Diese werden erforderlich, wenn man außerhalb von AWS beispielsweise in einer lokalen Entwicklungsumgebung autorisierten Zugriff auf dessen Ressourcen haben möchte. Zusätzlich wird für die Nutzung vom Amazon Braket Service das Erstellen einer Rolle, der sogenannten *AmazonBraketJobsExecutionRole*, erforderlich. Dazu empfiehlt es sich einen hybriden Job über die Benutzeroberfläche in AWS zu erstellen, sodass die Rolle sowie deren Berechtigung von AWS angelegt wird. Außerdem muss ein Amazon S3 Bucket mit Berechtigungen auf Amazon Braket in der gleichen Region verfügbar sein, in der auch das Amazon Braket Programm ausgeführt bzw. der hybride Job erstellt wird. Dies ist für das Ablegen der Ausgabedatei erforderlich.

Amazon Braket Programme sollen unter Verwendung der Python Amazon-Braket-SDK geschrieben werden. Dabei sollten alle Amazon Braket Programme, welche zusammengefügt werden sollen, auf dem gleichen Simulator beziehungsweise QPU ausgeführt werden. Dies liegt daran, weil eine Umgebungsvariable angelegt wird, in der entweder die QPU oder der Simulator spezifiziert wird. Hierbei können Amazon Braket Programme in zwei Bestandteile gegliedert werden: (i) Abhängigkeiten und (ii) Programmlogik. Dabei muss die Programmlogik einen eindeutigen Startpunkt in Form einer Methoden mit dem Namen `execute` enthalten. Enthält die Programmlogik Zuweisungen wie zum Beispiel `foo = 3 * bar()`, muss diese in zwei Anweisungen `foo = bar(); foo = 3 * foo` aufgespalten werden. Andernfalls wird der Funktionsaufruf als Variable durch das RedBaron[1] Objekt gewertet und die Methode wird nicht im hybriden Programm enthalten sein. Außerdem sollten Brakets nicht untereinander verlinkt sein, wenn solche Abhängigkeiten bestehen,

[1] https://redbaron.readthedocs.io/en/latest

Listing 6.1 Verwenden des Boto3 Clients für Amazon Braket Programme

```
import boto3
from botocore.config import Config
my_config = Config(
        region_name='us-east-1',
    )
braket_client = boto3.client('braket', aws_access_key_id=access_key, aws_secret_access_key=
secret_access_key, config=my_config)
kwargs = {
       'action': '{"braketSchemaHeader": {"name": "braket.ir.jaqcd.program", "version": "1"},
          "results": [], "basis_rotation_instructions": [], "instructions": [{"type": "h", "
target": 0}]}',
       'deviceArn': 'arn:aws:braket:::device/quantum-simulator/amazon/sv1',
       'deviceParameters': '{"braketSchemaHeader": {"name": "braket.device_schema.simulators.
gate_model_simulator_device_parameters", "version": "1"}, "paradigmParameters": {"
braketSchemaHeader": {"name": "braket.device_schema.gate_model_parameters", "version": "1"}, "
qubitCount": 1}}',
        'outputS3Bucket': 'amazon-braket-us-east-1-123456789101',
        'outputS3KeyPrefix': 'tasks',
        'shots': 10
    }
response = braket_client.create_quantum_task(**kwargs)
```

deutet dies auf eine sequentielle Ausführung im Workflow hin. In Listing 6.1 wird ein Circuit erstellt, der ein Hadamard Gate auf dem ersten Qubit hat. Dabei wird der Circuit auf dem Simulator SV1 ausgeführt und die Ergebnisse werden im Bucket `amazon-braket-us-east-1-123456789101` abgelegt. In diesem Beispiel wird im Amazon Braket Programm der Boto3 Client für die Ausführung eines Circuits verwendet (siehe Listing 6.1). Die Variablen `access_key` und `secret_access_key` sollen als Methodenparameter übergeben werden. Durch die Einführung des CodeGuru Reviewer Secrets Detector[2] muss für diese Variablen eine spezielle Behandlung erfolgen, da der hybride Job die Login-Daten nicht als Eingabeparameter akzeptiert und einen Validierungsfehler verursacht. Aus diesem Grund werden die Variablen `access_key` und `secret_access_key` aufgespalten, sodass sie dem Amazon Braket Hybrid Jobs übergeben werden können. Enthalten die Amazon Braket Programme Abhängigkeiten, die nicht in den drei möglichen Umgebungen[3] der Amazon Braket Hybrid Jobs enthalten sind, muss der Nutzer ein Container-Image für den Amazon Braket Hybrid Jobs erstellen und dem Workflow als Eingabeparameter übergeben. Dadurch das sowohl Amazon Braket Programme als auch die Amazon Braket Hybrid Jobs durch die Python Amazon-Braket-SDK beschrieben werden können, sind die Abhängigkeiten kompatibel. Zum Schluss müssen sowohl bei der Nutzung von Amazon Braket sowie Amazon Braket Hybrid Jobs die Kosten für die Nutzung von Simulatoren, QPUs und Shots beachtet werden.

[2] https://docs.aws.amazon.com/codeguru/detector-library/index
[3] https://docs.aws.amazon.com/braket/latest/developerguide/braket-jobs-script-environment

Listing 6.2 Möglichkeit 1: Erstellen eines Amazon Braket Hybrid Jobs durch AwsQuantumJob

```
job = AwsQuantumJob.create(
        device="arn:aws:braket:::device/quantum-simulator/amazon/sv1",
        source_module = "s3://amazon-braket-us-east-1-123456789101/hybrid-jobs.tar.gz",
        entry_point = "hybrid_program:main",
        wait_until_complete = True,
        jobName='HybridJob'
    )
```

6.2 Amazon Braket Hybrid Jobs Handler

Damit die automatisierte Generierung von Amazon Braket Hybrid Jobs durch den entsprechenden Handler möglich ist, wird der Qiskit Runtime Handler um die providerspezifischen Anforderungen von AWS erweitert. Dabei können Amazon Braket Hybrid Jobs auf drei verschiedene Weisen erstellt werden, wobei für dieses Szenario weder das Jupyter Notebook noch das manuelle Erstellen in der Amazon Braket Konsole relevant sind. Aus diesem Grund wird nur die Umsetzung von Amazon Braket Hybrid Jobs durch das Verwenden der Python Amazon-Braket-SDK betrachtet. Dabei definiert die SDK zwei Möglichkeiten, um Amazon Braket Hybrid Jobs auf AWS hochzuladen und auszuführen. Die erste Möglichkeit ist in Listing 6.2 zu sehen. Dazu werden die Parameter `device`, `source_module`, `entry_point` , `wait_until_complete` und `jobName` spezifiziert. Der device Parameter ist der *Amazon Resource Name (ARN)* eines Simulators oder einer QPU. Das source_module gibt die S3-URI an, an dem sich das hybride Programm in Amazon S3 befindet. Hierbei muss es sich standardmäßig um eine tar.gz Datei handeln. Der entry_point beschreibt, welche Datei und welche Methode der Startpunkt der Ausführung sind. Der letzte Parameter gibt an, ob die Ausführung solange blockiert wird bis der hybride Job ausgeführt wird. Somit wird in Listing 6.3 die Datei hybrid_jobs, welche sich im Amazon S3 Bucket *amazon-braket-us-east-1-123456789101* befindet, aufgerufen und der Job mit dem Namen *HybridJob* im Simulator SV1 ausgeführt. Der Start der Ausführung ist die `main` Methode der Datei *hybrid_program*. Es werden alle Logging-Einträge in Amazon Cloudwatch gesammelt. Außerdem ist die Ausführung des hybriden Jobs blockierend, sodass die Ausführung erst fortgesetzt wird, wenn der Job beendet oder fehlgeschlagen ist. Die zweite Möglichkeit ist deutlich länger, jedoch wird diese erforderlich, da das Konzept in das MODULO Framework integriert werden soll und damit eine Authentifizierung für AWS durch den Boto3 Client erforderlich wird. Dazu werden der access_key und der secret_access_key erforderlich. Zusätzlich kann die Region spezifiziert werden für die der Client gültig ist. Hierbei ist zu beachten, dass der Amazon S3 Bucket sowie die QPUs in der gleichen Region verfügbar sein müssen wie die vom Client. Somit ist es nicht möglich den Client für die Region us-west-2 zu authentifizieren, wenn beispielsweise der Lucy[4] Quantencomputer verwendet werden soll, der nur in eu-west-2 zugänglich ist. Der Parameter `algorithmSpecification` definiert, welche Datei und in welcher Umgebung der hybride Job ausgeführt werden soll. In dem Parameter `deviceConfig` ist die ARN der QPU oder des Simulators enthalten, auf dem der hybride Job ausgeführt wird. Die `hyperparameter` können für das Training von Machine-Learning Algorithmen verwendet werden oder als Eingabeparameter, da sie Schlüssel-Wert Paare sind. Somit hat der hybride Job direkten Zugriff auf die Werte und daher wird der Parameter `inputDataConfig` nicht verwendet. Dabei enthält dieser den Verweis auf die

[4] https://aws.amazon.com/de/braket/quantum-computers/oqc

Eingabedaten in Amazon S3. Durch `instanceConfig` werden die klassischen Ressourcen sowie deren Größe spezifiziert. Der `jobName` ist die Spezifikation eines eindeutigen Amazon Braket Hybrid Jobs Namens. Der Parameter `outputDataConfig` gibt an, wo die Ergebnisse des hybriden Jobs in Amazon S3 abgelegt werden sollen. Der letzte Parameter ist der Verweis auf die Rolle in AWS, welche über die Autorisierung zum Starten und Verwalten von Amazon Braket Hybrid Jobs entscheidet.

Durch beide Möglichkeiten wird durch die SDK eine Amazon Braket Hybrid Jobs Instanz erstellt, welche die QPU reserviert sowie die klassischen Ressourcen nach Bedarf provisioniert. Damit aus den Amazon Braket Programmen ein Amazon Braket Hybrid Jobs erstellt wird, werden im Handler neue Templates definiert sowie die Behandlung von Eingabe- und Ausgabeparametern adaptiert. Dazu wird ein amazon-braket-hybrid-job Template definiert, welches aus einer main Methode und Paketimporten zum Speichern der Ausgabeparameter besteht. Wie in Abschnitt 4.2 erwähnt, ist ein Amazon Braket Hybrid Jobs valide, wenn dieser aus den notwendigen Importen, Startpunkt und entsprechender Programmlogik besteht. Dadurch dass die main Methode bereits vor der Programmgenerierung feststeht, wird diese als eindeutiger Startpunkt für die spätere Ausführung festgelegt. Außerdem werden schrittweise die Importe sowie die jeweilige Programmlogik aus den verschiedenen Amazon Braket Programmen in das Template kopiert, sodass am Ende ein valider Amazon Braket Hybrid Jobs entsteht. Das Hochladen des Amazon Braket Hybrid Jobs erfolgt durch den Boto3 Client, indem das generierte Programm im jeweiligen Amazon S3 Bucket hochgeladen wird, dann wird der hybride Job erstellt und zum Schluss werden die Ausgabeparameter aus dem Amazon S3 Bucket extrahiert. Wie zuvor beschrieben, muss das hybride Programm in einem tar.gz Datei hochgeladen werden, sodass die JSON Datei ebenfalls in einer tar.gz Datei abgespeichert wird. Aus diesem Grund wird die Extrahierung im polling-agent dazu auch ergänzt.

6.3 Erweiterung des QuantME Transformation Frameworks

Das Transformation Framework wurde um den `AWSRuntimeHandler` erweitert, welcher für die Kommunikation mit dem Amazon Braket Hybrid Jobs Handler verantwortlich ist. Dazu wurde die Konfiguration des Modelers adaptiert, sodass ein neuer Endpoint für den Handler existiert. Mittels einer HTTP POST Anfrage werden dem Handler die Parameter `beforeLoop, afterLoop, requiredPrograms` und `loopCondition` übergeben. Dabei legt die Flask-Anwendung eine Task an die Redis Queue. Anschließend erhält das QuantME Transformation Framework die Location zurück, an dem sich das hybride Programm später befindet. Das QuantME Transformation Framework beginnt an der Location abzufragen, ob das generierte hybride Programm bereits existiert und wiederholt dies solange bis das hybride Programm abrufbar ist. Parallel überwacht der Redis Queue Worker die Redis Queue und nimmt eine Task aus der Queue. Daraufhin beginnt er mit der Programmgenerierung, die bereits im vorherigen Abschnitt beschrieben wurde. Wenn die Generierung erfolgreich war, wird das hybride Programm auf der Location hochgeladen. Aus diesem Grund beendet das QuantME Transformation Framework die Abfrage und beginnt mit der Winery zu interagieren, welches eine Modellierungsumgebung für TOSCA-basierte Cloud Anwendungen ist [KBBL13]. Hierbei werden Software- und Hardwarekomponenten basierend auf ihrer Beziehung zueinander in einem Graph angeordnet. Dabei ist jede Komponente ein *Node Template* und jede Beziehung ist ein *Relationship Template* [OAS13]. Die Implementierung der Node Template kann durch *deployment artifacts* beschrieben werden. Für die hybride Programmgenerierung wird ein neues deployment artifact angelegt, welches das hybride Programm als Implementierung für ein Node Template enthält. Anschließend wird eine neue Version der *CSAR AWSRuntimeAgentService*

Listing 6.3 Möglichkeit 2: Erstellen eines Amazon Braket Hybrid Jobs durch Boto3

```
client = boto3.client('braket', aws_access_key_id=access_key, aws_secret_access_key=
secret_access_key, config= my_config)
response = client.create_job(
    algorithmSpecification={
        'containerImage': {
            'uri': '292282985366.dkr.ecr.us-east-1.amazonaws.com/amazon-braket-base-jobs:1.0-
                    cpu-py37-ubuntu18.04'
        },
        'scriptModeConfig': {
            'compressionType': 'GZIP',
            'entryPoint': 'hybrid\_program:main',
            's3Uri': 's3://amazon-braket-us-east-1-123456789101/hybrid-jobs.tar.gz'
        }
    },
    deviceConfig={
        'device': 'arn:aws:braket:::device/quantum-simulator/amazon/sv1'
    },
    hyperParameters={}
    inputDataConfig=[],
    instanceConfig={
        'instanceCount': 1,
        'instanceType': 'ml.m5.large',
        'volumeSizeInGb': 1
    },
    jobName='HybridJob',
    outputDataConfig={
        's3Path': 's3://amazon-braket-us-east-1-123456789101/jobs/HybridJob/'
    },
    roleArn='arn:aws:iam::109876543210:role/service-role/AmazonBraketJobsExecutionRole'
)
```

erstellt, bei der das Deploymentartifakt durch das hybride Programm ersetzt wird. Die CSAR enthält alle für das Deployment benötigten Informationen wie beispielsweise den Graph oder die Artefakte, sodass CSARs zwischen verschiedenen TOSCA-Runtimes portabel sind. Zum Schluss wird der gesamte Optimierungskandidat durch einen Task ersetzt und der Sequenzfluss wird angepasst.

Neben dem Umschreiben von hybriden Programmen in Quanten Workflows, ist ebenfalls das Deployment und die Ausführung dieser hybriden Programme möglich. Dazu wird die Camunda REST API genutzt, um Variablen der Workflowinstanz abzuspeichern. Zudem wird der Boto3 Client verwendet, um die Authentifizierung für AWS zu ermöglichen. Zudem wird jedes hybride Programm in Amazon S3 hochgeladen sowie die Ausgabeparameter aus Amazon S3 extrahiert.

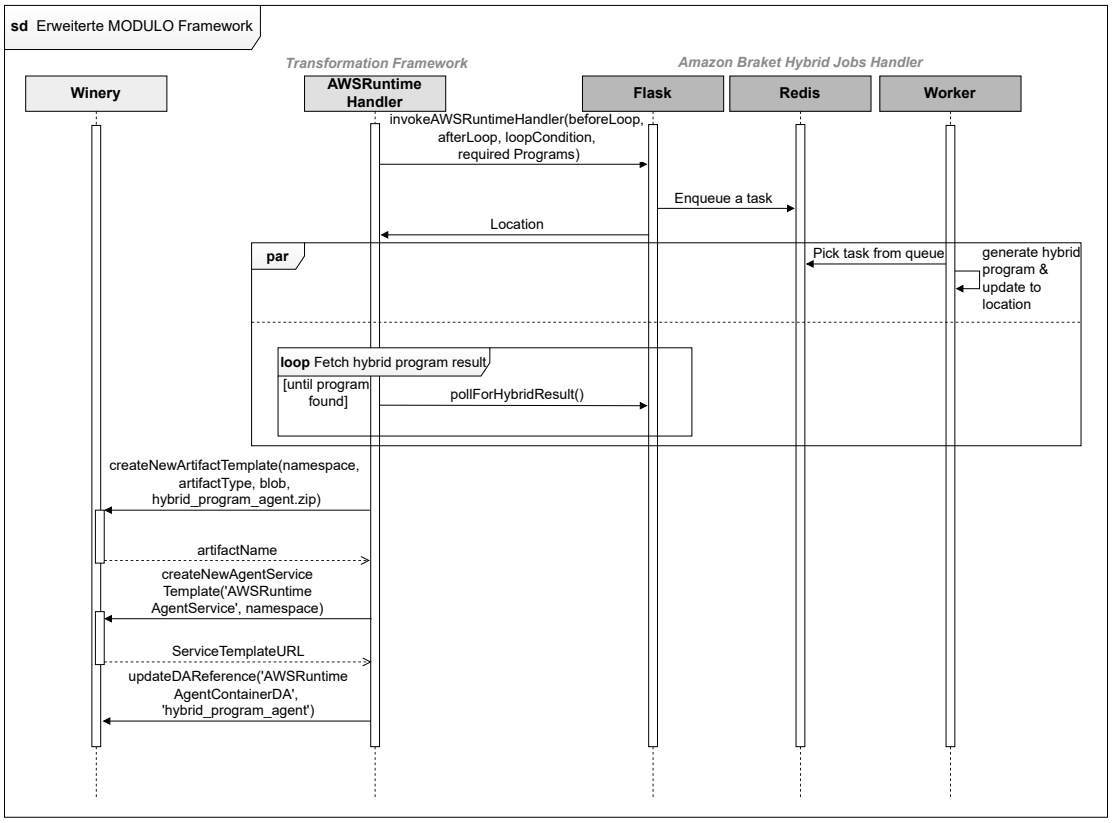

Abbildung 6.1: Kommunikation der Komponenten des MODULO Frameworks zur Erstellung eines hybriden Programms

7 Validierung des Konzepts und der Implementierung

In diesem Kapitel wird das Konzept und die prototypische Implementierung validiert. Dabei werden die in Abschnitt 4.2 definierten Anforderungen überprüft und auf ihre Erfüllbarkeit bewertet. Nachfolgend wird anhand eines beispielhaften Szenarios gezeigt, wie sich die Ausführung von Quanten Workflows durch das vorgestellte Konzept verbessert.

7.1 Überprüfung der Anforderungen

Dieser Abschnitt stellt die Validierung der Anforderungen dar. Die Bewertung der Anforderungen auf ihre Erfüllbarkeit ist in Tabelle 7.1 abgebildet. In der ersten Spalte werden die Anforderung aus Abschnitt 4.2 sowie eine kurze Beschreibung eingefügt. Die zweite Spalte zeigt das Resultat der Validierung. Ein Haken bedeutet, dass die Anforderung erfüllt wurde. Ist kein Haken vorhanden, konnte die Anforderung nicht erfüllt werden.

Das grundsätzliche Ziel der Arbeit besteht darin die automatische Generierung von Amazon Braket Hybrid Jobs zu ermöglichen. Dies konnte durch die Entwicklung des Amazon Braket Hybrid Jobs Handlers umgesetzt werden. Dazu wird in einem 3-Phasenmodell (siehe Abschnitt 5.1) ein valides Amazon Braket Hybrid Jobs erstellt. Somit ist die Anforderung A1 als auch A2 erfüllt.

Zusätzlich wurde das entwickelte Konzept in das MODULO Framework integriert, sodass neben der Qiskit Runtime auch Amazon Braket Hybrid Jobs verwendet werden können. Dadurch können Optimierungskandidaten für die Amazon Braket Hybrid Jobs erkannt, umgeschrieben und ausgeführt werden. Somit profitieren die Workflows von den Vorteilen der hybriden Runtime, sodass sich die Ausführungszeit der Workflows verkürzt (A4).

Anforderung	**Erfüllt?**
A1 (Generierung von Amazon Braket Hybrid Jobs)	✓
A2 (Prototypische Implementierung)	✓
A3 (Integrierung in das MODULO Framework)	✓
A4 (Evaluation der Ausführungszeit von Quanten Workflows)	✓

Tabelle 7.1: Bewertung der Anforderungen der Arbeit auf ihre Erfüllbarkeit

7.2 Anwendungsszenario

In diesem Abschnitt wird das Anwendungsszenario beschrieben, welches zur Verifikation und Anwendbarkeit des neuen Konzepts herangezogen wird. Dazu wird ein Quanten Workflow modelliert, welcher den VQE realisiert. Da es sich um einen variationellen Algorithmus handelt, welcher iterativ die Lösung bestimmt, gilt die Schleife als ein Optimierungskandidat im Workflow. Dieser Kandidat wird in ein hybrides Programm umgeschrieben.

In Abbildung 7.1 ist der Workflow zur Berechnung des kleinsten Eigenwerts einer Matrix zu sehen. Der Workflow wird durch Übergabe der Eingabeparameter `access_key`, `secret_access_key`, `roleArn`, `bucketName` sowie den Link zur Matrix und die QPU auf dem der Quanten Anteil ausgeführt werden soll, gestartet. Im ersten Service Task erfolgt die Initialisierung für den VQE-Algorithmus, indem die Parameter für die Quanten Ausführung bestimmt werden. Außerdem wird die Matrix von der URL abgefragt, sowie die Parameter der Pauli-Zerlegung für die Matrix definiert. Anschließend erfolgt der Eintritt in den Optimierungskandidaten, welcher aus einem Quantum Circuit Execution Task sowie einem Service Task besteht. Im Quantum Circuit Execution Task wird der parametrisierte Ansatz erstellt, der optimiert werden soll. Dabei wird für jeden Pauli-String bzw. jedes Quantenmodul der jeweilige Eigenwert bestimmt. Im nächsten Schritt erfolgt die Addition der Eigenwerte als auch die Adaption bzw. das Bestimmen der Parameter für die nächste Iteration. Dazu empfiehlt sich der Einsatz von bewährten klassischen Optimierungsmethoden wie beispielsweise der Powell Methode [Pow70] oder des Nelder Mead Algorithmus [NM65]. Bei Nelder Mead handelt es sich um eine numerische Methode, welche das Minimum beziehungsweise Maximum einer multidimensionalen Funktion f bestimmt [NM65]. Für unser Anwendungsszenario soll das Minimum des Eigenwerts berechnet werden, welcher durch einen parametrisierten Ansatz f repräsentiert wird. Dazu wird im Algorithmus für eine n-dimensionale Funktion ein Simplex mit $n + 1$ Eckpunkten definiert. Beispielsweise ist für eine 2-dimensionale Funktion der Simplex ein Dreieck. In der ersten Iteration erhält der Algorithmus die Initialisierungswerte *u, v, w* aus dem ersten Service Task und ordnet diese nach den Ergebnissen der Erwartungswerte. Sei $f(u) < f(v) < f(w)$ das Ergebnis der Sortierung, dann wird einer der vier Phasen ausgeführt: *Reflection*, *Expansion*,

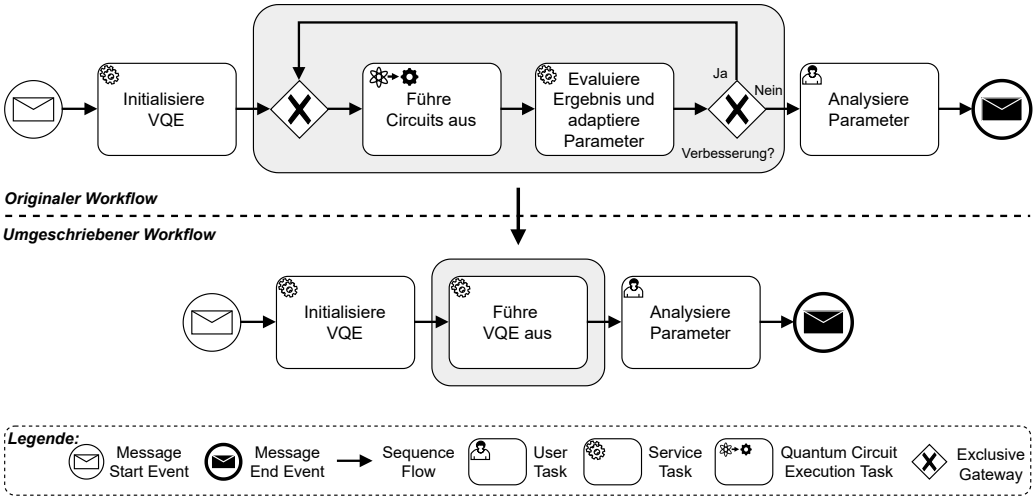

Abbildung 7.1: Quanten Workflow am Beispiel von VQE mit einer hybriden Schleife (adaptiert von [WBBL22])

Listing 7.1 Möglichkeiten zum Erstellen von Circuits

```
#### Option 1
from braket.circuits import Circuit
my_circuit = Circuit().h(range(1))
print(my_circuit)

#### Option 2
from braket.circuits import Gate, Instruction
circ = Circuit([Instruction(Gate.H(), 1)])
print(circ)
```

Contraction und *Shrink*. In der Reflection Phase wird der schlechteste Wert w am Mittelpunkt zwischen den anderen beiden Werten gespiegelt und man erhält einen Punkt r. Ist $f(r)$ besser als $f(w)$, wird w durch r ersetzt. In der Expansion Phase wird überprüft, ob eine Verbesserung entsteht, wenn in die Richtung des gespiegelten Werts r expandiert wird. Hingegen wird in der Contraction Phase der Wert w verworfen, wenn der Wert am Mittelpunkt zwischen den beiden anderen Wert besser ist als der am Punkt w. Die Shrink Phase verkleinert das Simplex in Richtung des kleinsten Punkts u. Dadurch erzielt der Nelder Mead Algorithmus iterativ immer eine Verbesserung in eine bestimmte Richtung, sodass das Minimum gefunden wird. Der Workflow terminiert, wenn keine Verbesserung mehr basierend auf einem Schwellwert eintritt. Jedoch ist anzumerken, dass die Konvergenz des Nelder Mead Algorithmus von der Wahl der Initialisierungswerte abhängt.

Der Optimierungskandidat aus Abbildung 7.1 wird im QuantME Transformation Framework erkannt, da iterativ klassische und Quanten Anteile ausgeführt werden. Wenn es sich hierbei um valide Amazon Braket Programme handelt, kann der Kandidat aktuell nicht für die Qiskit Runtime umgeschrieben werden. Da jede Runtime ihre eigenen proprietären Schnittstellen enthält sowie unterschiedliche Programmierkonstrukte enthalten, gibt es kein 1-zu-1 Mapping zwischen den Runtimes. Somit müsste jeder Circuit sowie dessen Aufruf durch die jeweils anderen Schnittstellen ersetzt werden. In Listing 7.1 wurde der gleiche Circuit aus Listing 6.1 auf zwei weitere erstellt. Dadurch sieht man, welche Komplexität entstehen würde, wenn der Qiskit Runtime Handler um diese Möglichkeiten ergänzt werden müsste. Außerdem unterscheidet sich der dritte Ansatz noch per QPU beziehungsweise Simulator. Zusätzlich müsste die zuvor gewählte QPU von den Amazon Braket Programmen durch eine geeignete QPU von IBMQ ersetzt werden, auf die der Circuit ausgeführt werden kann. Aus diesem Grund ist dieser Ansatz zeitintensiv und fehleranfällig. Daher kann der Kandidat für die Amazon Braket Hybrid Jobs umgeschrieben werden und auch in dieser hybriden Runtime ausgeführt werden. Der umgeschriebene Workflow besteht nur noch aus drei Tasks, wobei die Service Task zur Initialisierung sowie die User Task unverändert bleiben. Der neue Task lädt das entsprechende hybride Programm in Amazon S3 hoch, führt das hybride Programm aus und gibt die Ausgabeparameter wieder an den Workflow zurück.

8 Evaluation

Dieses Kapitel beinhaltet die Evaluation des in dieser Arbeit entwickelten Konzepts und der darauf basierenden prototypischen Implementierung. Zunächst wird die Testumgebung vorgestellt, auf der die jeweilige Ausführung stattfindet. Anschließend wird betrachtet, wie sich die Zeit zum Umschreiben mit steigender Anzahl an Aktivitäten im Workflow verhält. Zum Schluss wird die Ausführungszeit mit und ohne Einsatz der hybriden Runtime verglichen.

8.1 Testumgebung

Die Testumgebung besitzt folgende Eigenschaften:

Betriebssystem Windows 11 Home 10.0.22000 Build 22000

Prozessor Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHZ. 1190 MHz.

RAM 16 GB

8.2 Analyse der Programmgenerierung

In diesem Abschnitt wird die Zeit zum Umschreiben von Amazon Braket Programmen zu Amazon Braket Hybrid Jobs festgehalten. Dazu wurde die Testumgebung aus dem vorherigen Abschnitt sowie die Implementierung des Amazon Braket Hybrid Jobs Handlers aus Kapitel 6 genutzt. Dabei soll der Einfluss von Kandidaten und deren Größe auf die Zeit für die Generierung der hybriden Programme untersucht werden.

Für die Analyse der Ausführung wird die Zeit zum Umschreiben von Optimierungskandidaten für sieben verschiedene Workflows, welche in Tabelle 8.1 dargestellt sind, gemessen. Die Zeit wird hier aus dem Median der 20 Durchläufe ermittelt, wobei der Median aufgrund seiner Robustheit gegen Ausreißer gewählt wurde. Die Eigenschaften der Workflows wie die Anzahl an Kandidaten und die durchschnittliche Anzahl an Aktivitäten pro Kandidat haben Einfluss auf das Verhalten der Ausführungszeit für das Umschreiben eines Kandidaten. Die Anzahl an Kandidaten ist dabei analog zur Anzahl an zu generierenden hybriden Programme. Desto mehr Aktivitäten in diesen Kandidaten vorhanden sind, desto mehr Zeilen an Code müssen untersucht und zusammengefasst werden. Zu beachten ist, dass die Aktivitäten des Workflows in Python geschrieben sind und ungefähr 300-400 Zeilen Code umfassen. Der Workflow mit der ID 1 entspricht dem Anwendungsszenario aus Kapitel 7. Dieser enthält vier Aktivitäten sowie einen Optimierungskandidaten. Das Umschreiben dieses Kandidaten benötigt ca. 101 Sekunden. Im Vergleich dazu lässt sich trotz ergänzter Aktivitäten im Workflow mit ID 2 kaum ein Unterschied in der Zeit zur Generierung des hybriden Programms erkennen. Die Workflows mit ID 3 und ID 4 werden um jeweils einen Optimierungskandidaten

erweitert. Der Workflow mit ID 3 benötigt ca. 201s, während Workflow mit ID 4 ca. 298s braucht. Hierbei lässt sich ein lineares Wachstum der Zeit zur Generierung und zum Umschreiben erkennen, welches auf die sequentielle Verarbeitung der Kandidaten zurückzuführen ist [WBBL22]. Derzeit ist jedoch keine parallele Generierung der hybriden Programme für die Kandidaten möglich um die Zeit für das Umschreiben zu verkürzen. Außerdem wurde die Ausführung für die Workflows mit ID 5-7 so verändert, dass jeder Kandidat nun vier Aktivitäten pro Kandidat enthält. Zu erkennen ist, dass die Workflows ein ähnliches Wachstumsverhalten aufweisen wie die Workflows mit ID 2-4. Jedoch benötigen die jeweils korrespondierenden Workflows von ID 2-4 mehr Zeit. Dies lässt sich auf die steigende Anzahl an Codezeilen sowie der wachsenden Komplexität für das Umschreiben zurückführen.

Abschließend lässt sich sagen, dass die Anzahl an Aktivitäten kaum einen Einfluss auf die Zeit zur Generierung und zum Umschreiben hat. Das Wachstum für die Dauer des Umschreibens ist linear zur Anzahl an Kandidaten. Jedoch lässt sich eine erhöhte Dauer bei steigender Anzahl von Aktivitäten pro Kandidat erkennen.

ID	# Aktivitäten	# Kandidaten	∅ Aktivitäten pro Kandidat	∅ Dauer Umschreiben
1	4	1	2	100.99 s
2	20	1	2	101.02 s
3	20	2	2	201.20 s
4	20	3	2	298.44 s
5	20	1	4	235.98 s
6	20	2	4	476.51 s
7	20	3	4	703.12 s

Tabelle 8.1: Ausführungszeit für das Umschreiben des Workflows

8.3 Vergleich der Ausführungszeit von Workflows

Dieser Abschnitt befasst sich mit der Ausführungszeit des Workflows aus Kapitel 7. Dabei wird die Zeit des originalen sowie des umgeschriebenen Workflows gemessen und verglichen. Der Workflow wird auf drei verschiedenen Quantensimulatoren ausgeführt. Der umgeschriebene Workflow nutzt dabei die Amazon Braket Hybrid Jobs, während der originale Workflow aus Amazon Braket Programmen besteht.

Die Ergebnisse der Ausführungszeit des Workflows für verschiedene Geräte sind in Tabelle 8.2 zusammengefasst. In Spalte 1 sind die verschiedenen Quantenressourcen aufgelistet. Die Spalte 2 und 3 umfasst die Dauer des originalen sowie des umgeschriebenen Workflows. Die Dauer entspricht dem Median von zehn Ausführungen des jeweiligen Workflows. Der Simulator SV1 braucht 3605s für den originalen Worklow. Im Vergleich dazu werden für den umgeschriebenen Workflow 915s weniger benötigt, was einer Ersparnis von ca. 25% entspricht. Dieses Verhalten ist analog für die Simulatoren TN1 und DM1 zu erkennen.

Aus den Daten der Evaluation geht hervor, dass die Zeit für das Ausführen des Workflows bei der Verwendung des neuen Konzepts deutlich reduziert wird. Dies lässt sich durch die Ausführung der nah beinander liegenden Ressourcen erklären sowie durch den Prioritätszugriff auf die Warteschlange und den daraus resultierenden minimierten Latenzen.

Simulator	∅ **Dauer originaler Workflow**	∅ **Dauer umgeschriebener Workflow**
Simulator SV1	3605 s	2690 s
Simulator TN1	3876 s	2788 s
Simulator DM1	3904 s	2854 s

Tabelle 8.2: Vergleich der Ausführungszeit des originalen und umgeschriebenen Workflows

9 Zusammenfassung und Ausblick

Die zunehmende Forderung nach Sicherheit und Effizienz setzt neue Paradigmen wie das Quantencomputing voraus. Dabei nutzt das Quantencomputing die Vorteile der Quantenmechanik und Numerik. Um Quantencomputer für die Allgemeinheit verfügbar zu machen, werden diese durch Cloud Provider wie Amazon oder IBM angeboten. Es können somit in Amazon beispielsweise Amazon Braket Programme erstellt werden, die auf Python basieren. Jedoch sind Quantencomputer sogenannte NISQ Computer, welche in ihrer Anzahl an Qubits beschränkt und fehleranfällig sind. Deshalb sollen nur Schritte, die auf einem klassischen Computer nicht oder langsamer gelöst werden können, auf einem Quantencomputer ausgeführt werden. Aus diesem Grund werden hybride Algorithmen benötigt, welche die Ausführung auf dem Quantencomputer auf die essentiellen Schritte reduzieren. Die anderen Schritte, die sogenannte Vorverarbeitung und Nachbearbeitung, werden auf klassischen Ressourcen ausgeführt. Um eine Orchestrierung dieser Schritte zu ermöglichen, wurde das MODULO Framework eingeführt. Somit können Quanten Workflows in BPMN mit QuantME Modellierungskonstrukten visualisiert und ausgeführt werden. Jedoch entstehen hier weiterhin Latenzen durch den Austausch der Daten der geographisch entfernten klassischen Computer und Quantencomputer. Bei Optimierungsalgorithmen sogenannten variationellen Algorithmen wie QAOA und VQE werden mehrere Iterationen durchgeführt, wodurch weitere Latenzen entstehen.

Aus diesem Grund wurde das QuantME Transformation Framework um die Kommunikation mit hybriden Runtime Handlern erweitert. Durch die Verwendung von hybriden Runtimes können Iterationen von klassischen und Quanten Teilen durch ein hybrides Programm ersetzt und ausgeführt werden. Der Vorteil von hybriden Runtimes ist es, dass der Provider die CPU und QPU nah beieinander wählt und somit geringere Latenzen entstehen. Diese Funktionalität konnte bisher nur für Qiskit Programme genutzt werden. Damit dies nun auch für Amazon Braket Programme ausführbar ist, wurde die Implementierung um den Amazon Braket Hybrid Jobs Handler ergänzt. Somit wird automatisiert ein hybrides Programm generiert, welches die Amazon Braket Programme zusammenfasst. Die Generierung gliedert sich in Extraktion, Filterung der Eingabeparameter sowie Bestimmen der Ausgabeparameter und das Zusammenfügen in ein hybrides Programm. Die Herausforderungen für die Generierung sind das Übergeben der Parameter sowie das Auslesen der Parameter, da die Ergebnisse in S3 hochgeladen werden müssen.

Außerdem erfolgte eine Validierung des Konzepts durch das Anwendungsszenario, indem der aus Braket Programmen bestehende VQE Algorithmus zu einem hybriden Algorithmus zusammengefasst wird. Nachfolgend wurde die Evaluation eines Quanten Workflows mit unterschiedlicher Anzahl an Aktivitäten und Ersetzungskandidaten durchgeführt. Hierbei wurde die Performanz für die Ausführung des Workflows mit und ohne hybrider Runtime verglichen. Zudem wurden für die Ausführungen mit hybrider Runtime unterschiedliche Simulatoren verwendet. Das Ergebnis zeigt, dass sich die Ausführungszeit für hybride Algorithmen mit Runtime verbessert.

Ausblick

Die in dieser Arbeit vorgestellte Methode zur automatisierten Generierung von Amazon Braket Hybrid Jobs verbessert die Ausführungszeit von Quanten Workflows, welche Optimierungskandidaten enthalten. Dabei ist die Auswahl von hybriden Runtimes auf die Qiskit Runtime und Amazon Braket Hybrid Jobs beschränkt. Eine mögliche Erweiterung wäre es zukünftige hybride Runtimes hinzuzufügen sowie die Methode mit weiteren Quantencomputern zu testen. Außerdem können die im Mai 2022 von AWS vorgestellten eingebetteten Simulatoren, welche eine Verbesserungen der Laufzeit versprechen, untersucht werden. Zusätzlich kann die Implementierung um den seit Juni 2022 verfügbaren Qiskit-Braket-Provider [Sul22] ergänzt werden. Somit könnten Qiskit Programme durch den Handler in einen Amazon Braket Hybrid Jobs umgeschrieben werden. Dabei kann untersucht werden, ob ein Speedup erzeugt wird sowie AWS spezifische QPUs genutzt werden.

Literaturverzeichnis

[Ama20] Amazon. *Amazon Braket Hybrid Job Developerguide*. (2020). URL: https://docs.aws.amazon.com/braket/latest/developerguide/braket-jobs-works (zitiert auf S. 20).

[Bar22] J. Barzen. „From Digital Humanities to Quantum Humanities: Potentials and Applications“. In: *Quantum Computing in the Arts and Humanities: An Introduction to Core Concepts, Theory and Applications*. Hrsg. von E. R. Miranda. Cham: Springer International Publishing, (2022), S. 1–52. DOI: 10.1007/978-3-030-95538-0_1 (zitiert auf S. 15).

[CAB+21] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, et al. „Variational quantum algorithms“. In: *Nature Reviews Physics* 3.9 (2021), S. 625–644. DOI: 10.1038/s42254-021-00348-9 (zitiert auf S. 20).

[Dir39] P. A. M. Dirac. „A new notation for quantum mechanics“. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 35.3 (1939), S. 416–418. DOI: 10.1017/S0305004100021162 (zitiert auf S. 20).

[FGG14] E. Farhi, J. Goldstone, S. Gutmann. *A Quantum Approximate Optimization Algorithm*. (2014). DOI: 10.48550/ARXIV.1411.4028 (zitiert auf S. 18).

[Gro96] L. K. Grover. *A fast quantum mechanical algorithm for database search*. (1996). DOI: 10.48550/ARXIV.QUANT-PH/9605043 (zitiert auf S. 17).

[Hom18] M. Homeister. „Grundlagen – Anwendungen – Perspektiven“. In: *Quantum Computing verstehen*. Springer Vieweg Wiesbaden, (2018). DOI: 10.1007/978-3-658-22884-2 (zitiert auf S. 17).

[HTC18] C. Havenstein, D. Thomas, S. Chandrasekaran. „Comparisons of Performance between Quantum and Classical Machine Learning“. In: *SMU Data Science Review*. (2018). DOI: 10.13140/RG.2.2.20353.40801 (zitiert auf S. 15, 17).

[JAA+18] A. J., A. Adedoyin, J. Ambrosiano, P. Anisimov, A. Bärtschi, W. Casper, et al. „Quantum Algorithm Implementations for Beginners“. In: *arXiv*. (2018), S. 9. DOI: 10.48550/ARXIV.1804.03719 (zitiert auf S. 17).

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. „Winery — A Modeling Tool for TOSCA-Based Cloud Applications“. In: *Proceedings of the 11th International Conference on Service-Oriented Computing - Volume 8274*. ICSOC 2013. Berlin, Germany: Springer-Verlag, (2013), S. 700–704. ISBN: 9783642450044. DOI: 10.1007/978-3-642-45005-1_64. URL: https://doi.org/10.1007/978-3-642-45005-1_64 (zitiert auf S. 36).

[KBG+16] K. Képes, U. Breitenbücher, S. Gómez Sáez, J. Guth, F. Leymann, M. Wieland. „Situation-Aware Execution and Dynamic Adaptation of Traditional Workflow Models“. In: *Proceedings of the 5th European Conference on Service-Oriented and Cloud Computing (ESOCC)*. Springer International Publishing, (2016), S. 69–83. DOI: 10.1007/978-3-319-44482-6_5 (zitiert auf S. 25).

[KMT+17] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, J. M. Gambetta. „Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets“. In: *Nature* 549 (2017), S. 242–246 (zitiert auf S. 15).

[KTP+20] P. Karalekas, N. Tezak, E. Peterson, C. Ryan, M. Silva, R. Smith. „A quantum-classical cloud platform optimized for variational hybrid algorithms“. In: *Quantum Science and Technology* 5.2 (2020), S. 2–4. DOI: 10.1088/2058-9565/ab7559 (zitiert auf S. 19).

[LDUD10] M. La Rosa, M. Dumas, R. Uba, R. Dijkman. „Merging business process models“. In: *OTM Conferences (1)* (2010), S. 96–113 (zitiert auf S. 25).

[Ley21] F. Leymann. „Wenn nicht jetzt, wann dann?“ In: Digitale Welt, (2021). DOI: 10.1007/s42354-021-0333-9 (zitiert auf S. 18).

[LR00] F. Leymann, D. Roller. „Production Workflow: Concepts and Techniques“. In: *Prentice Hall PTR*. (2000) (zitiert auf S. 21, 22).

[MGKR15] N. Mundbrod, G. Grambow, J. Kolb, M. Reichert. „Context-Aware Process Injection: Enhancing Process Flexibility by Late Extension of Process Instances“. In: *OTM Conferences*. (2015). DOI: 10.1007/978-3-319-26148-5_8 (zitiert auf S. 25).

[NC10] M. A. Nielsen, I. L. Chuang. „Quantum Computation and Quantum Information: 10th Anniversary edition“. In: *Cambridge University Press*. (2010). DOI: 10.1017/CBO9780511976667 (zitiert auf S. 15, 17, 19).

[NM65] J. A. Nelder, R. Mead. „A Simplex Method for Function Minimization“. In: *Comput. J.* 7 (1965), S. 308–313 (zitiert auf S. 40).

[NSL+14] S. Nastic, S. Sehic, D.-H. Le, H.-L. Truong, S. Dustdar. „Provisioning Software-Defined IoT Cloud Systems“. In: *Proceedings - 2014 International Conference on Future Internet of Things and Cloud, FiCloud 2014* (2014), S. 288–295. DOI: 10.1109/FiCloud.2014.52 (zitiert auf S. 22, 27).

[OAS07] OASIS. „Web Services Business Process Execution Language (WS-BPEL) Version 2.0“. In: Organization for the Advancement of Structured Information Standards, (2007) (zitiert auf S. 21).

[OAS13] OASIS. *Topology and Orchestration Specification for Cloud Applications Version 1.0*. (2013). URL: https://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html (zitiert auf S. 36).

[OMG11] OMG. „Business Process Model and Notation (BPMN) Version 2.0“. In: Object Management Group, (2011) (zitiert auf S. 21).

[PMS+14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, J. L. O'Brien. „A variational eigenvalue solver on a photonic quantum processor“. In: *Nature Communications* 5.1 (2014). DOI: 10.1038/ncomms5213 (zitiert auf S. 15, 17, 18).

[Poc21] D. Poccia. *Introducing Amazon Braket Hybrid Jobs - Set Up, Monitor, and Efficiently Run Hybrid Quantum-Classical Workloads*. (2021). URL: https://aws.amazon.com/de/blogs/aws/introducing-amazon-braket-hybrid-jobs-set-up-monitor-and-efficiently-run-hybrid-quantum-classical-workloads (zitiert auf S. 15).

[Pow70] M. J. D. Powell. „A Survey of Numerical Methods for Unconstrained Optimization“. In: *SIAM Rev.* 12.1 (1970), S. 79–97. DOI: 10.1137/1012004 (zitiert auf S. 40).

[Pre18] J. Preskill. „Quantum Computing in the NISQ era and beyond“. In: *Quantum* 2 (2018), S. 79. DOI: 10.22331/q-2018-08-06-79 (zitiert auf S. 15, 17, 18).

[Qis21] Qiskit. *Qiskit Documentation*. (2021). URL: https://quantum-computing.ibm.com/lab/docs/iql/runtime (zitiert auf S. 15).

[RI16] M. Ripon, M. M. Islam. „An Overview on Quantum Computing as a Service (QCaaS): Probability or Possibility“. In: *International Journal of Mathematical Sciences and Computing* 2 (2016), S. 16–22. DOI: 10.5815/ijmsc.2016.01.02 (zitiert auf S. 18).

[Sho94] P. Shor. „Algorithms for quantum computation: discrete logarithms and factoring“. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. (1994), S. 124–134. DOI: 10.1109/SFCS.1994.365700 (zitiert auf S. 17).

[SKY06] S. Sun, A. Kumar, J. Yen. „Merging workflows: A new perspective on connecting business processes“. In: *Decision Support Systems* 42 (2006), S. 844–858. DOI: 10.1016/j.dss.2005.07.001 (zitiert auf S. 25).

[SS14] H. Singh, A. Sachdev. „The Quantum way of Cloud Computing“. In: *ICROIT 2014 - Proceedings of the 2014 International Conference on Reliability, Optimization and Information Technology* (2014), S. 397–400. DOI: 10.1109/ICROIT.2014.6798362 (zitiert auf S. 18).

[Sul22] J. Sullivan. *AWS*. (2022). URL: https://aws.amazon.com/de/blogs/quantum-computing/introducing-the-qiskit-provider-for-amazon-braket/ (zitiert auf S. 48).

[TM20] N. Taleb, E. Mohamed. „Cloud Computing Trends: A Literature Review“. In: *Academic Journal of Interdisciplinary Studies* 9 (2020), S. 91. DOI: 10.36941/ajis-2020-0008 (zitiert auf S. 15, 18).

[VBL+21] D. Vietz, J. Barzen, F. Leymann, B. Weder, V. Yussupov. „An Exploratory Study on the Challenges of Engineering Quantum Applications in the Cloud“. In: *Proceedings of the 2nd Quantum Software Engineering and Technology Workshop (Q-SET 2021) co-located with IEEE International Conference on Quantum Computing and Engineering (QCE21)*. CEUR Workshop Proceedings, (2021), S. 1–12. URL: http://ceur-ws.org/Vol-3008/paper1.pdf (zitiert auf S. 28).

[Vog11] M. Vogel. „Quantum Computation and Quantum Information, by M.A. Nielsen and I.L. Chuang“. In: *Contemporary Physics* 52 (2011), S. 604–605. DOI: 10.1080/00107514.2011.587535 (zitiert auf S. 18).

[WBBL22] B. Weder, J. Barzen, M. Beisel, F. Leymann. „Analysis and Rewrite of Quantum Workflows: Improving the Execution of Hybrid Quantum Algorithms“. In: *Proceedings of the 12th International Conference on Cloud Computing and Services Science (CLOSER 2022)*. SciTePress, (2022), S. 38–50. DOI: 10.5220/0011035100003200 (zitiert auf S. 25–28, 30, 31, 40, 44).

[WBL+20] B. Weder, J. Barzen, F. Leymann, M. Salm, D. Vietz. „The Quantum Software Lifecycle“. In: *Proceedings of the 1st ACM SIGSOFT International Workshop on Architectures and Paradigms for Engineering Quantum Software (APEQS 2020)*. ACM, (2020), S. 2–9. DOI: 10.1145/3412451.3428497 (zitiert auf S. 15).

[WBL21] B. Weder, J. Barzen, F. Leymann. „MODULO: Modeling, Transformation, and Deployment of Quantum Workflows“. In: *Proceedings of the 25th IEEE International Enterprise Distributed Object Computing Workshop (EDOCW 2021)*. IEEE Computer Society, (2021), S. 341–344. DOI: 10.1109/EDOCW52865.2021.00067 (zitiert auf S. 15, 22, 30).

[WBLS21] B. Weder, J. Barzen, F. Leymann, M. Salm. „Automated Quantum Hardware Selection for Quantum Workflows“. In: *Electronics* 10.8 (2021). URL: https://www.mdpi.com/2079-9292/10/8/984 (zitiert auf S. 21).

[WBLV21a] B. Weder, J. Barzen, F. Leymann, D. Vietz. „Quantum Software Development Lifecycle“. In: *arXiv*. (2021). DOI: 10.48550/ARXIV.2106.09323 (zitiert auf S. 18).

[WBLV21b] M. Weigold, J. Barzen, F. Leymann, D. Vietz. „Patterns for Hybrid Quantum Algorithms“. In: *Proceedings of the 15th Symposium and Summer School on Service-Oriented Computing (SummerSOC 2021)*. Springer International Publishing, (2021), S. 34–51. DOI: 10.1007/978-3-030-87568-8_2 (zitiert auf S. 18, 19).

[WBLW20] B. Weder, U. Breitenbücher, F. Leymann, K. Wild. „Integrating Quantum Computing into Workflow Modeling and Execution“. In: *Proceedings of the 13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020)*. IEEE Computer Society, (2020), S. 279–291. DOI: 10.1109/UCC48980.2020.00046 (zitiert auf S. 21, 22).

[WBLZ21] B. Weder, J. Barzen, F. Leymann, M. Zimmermann. „Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective“. In: *2021 IEEE International Conference on Web Services (ICWS)*. (2021), S. 1–13. DOI: 10.1109/ICWS53863.2021.00015 (zitiert auf S. 15, 23).

[ZHM14] M. Zemni, N. Hadj-Alouane, A. Mammar. „Business Process Fragments Behavioral Merge“. In: *OTM Conferences*. (2014). DOI: 10.1007/978-3-662-45563-0_7 (zitiert auf S. 25).

Alle URLs wurden zuletzt am 01. 09. 2022 geprüft.