Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Entwicklung einer Generischen Architektur für Quanten-Services in der Cloud

Ken Singer

Studiengang: Softwaretechnik

Prüfer/in: Prof. Dr. Dr. h. c. Frank Leymann

Betreuer/in: Daniel Vietz

Beginn am: 24. April 2019

Beendet am: 24. Oktober 2019

Kurzfassung

Quantenalgorithmen haben das Potenzial bestimmte Probleme deutlich schneller lösen zu können, als klassische Algorithmen. Daher kann es in Zukunft von Vorteil sein, klassische Anwendungen durch Quantenalgorithmen zu unterstützen. Für die Programmierung eines Quantenalgorithmus werden Frameworks benötigt, welche häufig nur für eine Programmiersprache verfügbar sind. Dadurch sind sie nicht in jede beliebige Anwendung integrierbar. Außerdem stellt jeder Hersteller von Quantencomputern ein eigenes Framework bereit, das sich unter anderem in der Syntax von den anderen Frameworks unterscheidet und ein anderes Paradigma bei der Konstruktion der Quantencomputer verfolgt. Dadurch gibt es Unterschiede in den Implementierungen eines Algorithmus, wenn er für verschiedene Provider entwickelt wird. Im Rahmen dieser Arbeit werden die existierenden Paradigmen und Frameworks untersucht. Die daraus resultierenden Erkenntnisse werden für die Konzeption einer Plattform verwendet, die Quantenalgorithmen verwaltet und sie als Quanten-Services bereitstellt, um die Algorithmen programmiersprachenunabhängig verwenden zu können und die Details zu Framework, Paradigma und Implementierung zu verbergen. Diese Arbeit spezifiziert Anforderungen für die Plattform bzw. Quanten-Services und entwirft eine Architektur. Darüber hinaus werden empfohlene Praktiken für die Umsetzung von Quanten-Services herausgearbeitet. Um das vorgestellte Konzept zu validieren, wird die Plattform prototypisch umgesetzt. Das Anwendungsbeispiel zeigt, dass die Plattform, unabhängig vom verwendeten Framework, Quanten-Services aus Quantenalgorithmen generiert. Die entstandenen Quanten-Services können in der Cloud bereitgestellt und programmiersprachenunabhängig von einer klassischen Anwendung verwendet werden.

Danksagung

Hiermit möchte ich mich gerne bei allen bedanken, die mich während meines Studiums und beim Bearbeiten dieser Masterarbeit unterstützt haben.

Ich bedanke mich bei meinem Betreuer Daniel Vietz, der mir die Bearbeitung dieses Themas ermöglicht hat. Die kompetente fachliche Unterstützung und das Feedback haben mir bei der Ausarbeitung sehr geholfen.

Außerdem möchte ich mich bei allen Korrekturlesern bedanken, die mir beim Auffinden von Fehlern geholfen und Verbesserungsvorschläge gemacht haben.

Ein besonderer Dank gilt meinen Eltern, die mich während meiner gesamten Studienzeit und darüber hinaus unterstützt haben.

Inhaltsverzeichnis

1 Einleitung **17**

2 Grundlagen **19**
2.1 Quantencomputer-Paradigmen 19
2.2 Quantencomputing 22
2.3 Services 27

3 Verwandte Arbeiten **29**
3.1 Plattform für Quantenalgorithmen 29
3.2 Modulare Compiler 29
3.3 Generierung von Schnittstellen 30

4 Quanten-Frameworks **31**
4.1 Qiskit 36
4.2 Cirq 36
4.3 Forest SDK 36
4.4 Strawberry Fields 37
4.5 Quantum Development Kit 37
4.6 Ocean 37
4.7 ProjectQ 38
4.8 PennyLane 38
4.9 XACC 39

5 Konzeption **41**
5.1 Problemstellung 41
5.2 Erkenntnisse 41
5.3 Anforderungen 42
5.4 Architektur 43
5.5 Empfohlene Praktiken 44

6 Implementierung **47**
6.1 Quantum Service Platform 47
6.2 Quanten-Services 48
6.3 Anwendungsbeispiel 49

7 Zusammenfassung und Ausblick **55**

Literaturverzeichnis **57**

Abbildungsverzeichnis

2.1 Verflechtung von Anionen 21
2.2 Polarisationsstrahlteiler 23
2.3 Verschränkung auf einem Quantencomputer 25
2.4 Spiegelung der Amplituden 26

4.1 Architektur der Frameworks 31
4.2 Hybride Berechnungen 38

5.1 Architektur der Plattform 44

6.1 Entity-Relationship-Diagramm der Datenbank 48
6.2 Paketdiagramm eines Quanten-Service 49
6.3 Sequenzdiagramm des Anwendungsbeispiels 50
6.4 Algorithmus-Formular in QUASP 50
6.5 Parameter-Formular in QUASP 51
6.6 Implementierung-Formular in QUASP 52
6.7 Algorithmen-Übersicht in QUASP 53
6.8 Implementierungs-Übersicht in QUASP 53
6.9 Benutzeroberfläche des Quanten-Service 54

Tabellenverzeichnis

4.1 Vergleichstabelle Frameworks 34
4.2 Vergleichstabelle Quantencomputer 35

5.1 Anforderungen Plattform 42
5.2 Anforderungen Quanten-Service 43

Verzeichnis der Codebeispiele

6.1 Qiskit-Implementierung des Shor-Algorithmus 51

Akronyme

API Application Programming Interface. 30

BQM Binary Quadratic Model. 37

Cbit klassisches Bit. 22

CLI Command-Line Interface. 56

CNOT Controlled-NOT. 20

DSL domänenspezifische Sprache. 31

HATEOAS hypermedia as the engine of application state. 27

KI künstliche Intelligenz. 36

OpenQASM Open Quantum Assembly Language. 29

QAaaS Quantum Algorithm as a Service. 17

QCaaS Quantum Computing as a Service. 17

QDK Quantum Development Kit. 31

QPS Quantum Programming Studio. 29

QUASP Quantum Service Platform. 47

Qubit Quantenbit. 17

Quil Quantum Instruction Language. 36

QVM Quantum Virtual Machine. 36

REST Representational State Transfer. 27

SaaS Software as a Service. 17

SOA Service-orientierte Architektur. 27

1 Einleitung

Aufwendige Berechnungen werden aktuell auf klassischen Systemen, wie Supercomputern oder in der Cloud ausgeführt. Diese Berechnungen können sehr lange dauern oder theoretisch so viel Zeit beanspruchen, dass eine Berechnung nicht praktikabel ist. Das liegt daran, dass klassische Computer Amdahl´s Law [HM08] unterliegen. Das Gesetz besagt, dass für klassische Berechnungen mit sequenziellem Anteil der Geschwindigkeitszuwachs durch zusätzliche Hardware nach oben hin beschränkt ist. Im Gegensatz dazu skalieren Quantencomputer in der Theorie sehr gut vertikal, indem weitere Quantenbits (Qubits) hinzugefügt werden. Jedes zusätzliche Qubit verdoppelt die Rechenleistung [SS14]. Dadurch sind sie theoretisch in der Lage bestimmte Probleme, wie z. B. Suchen [Gro96] oder Faktorisierung [Sho94], deutlich schneller zu lösen. Dementsprechend haben Unternehmen und Technologiekonzerne großes Interesse an Quantencomputern. Google, Microsoft, IBM und viele weitere forschen an Quantencomputern [Fin19] und Unternehmen wie Lockheed Martin und Google besitzen Quantencomputer von D-Wave [D-Wa].

Laut Singh und Sachdev [SS14] sind Quantencomputer sehr teuer, weshalb sie vorschlagen die Kapazitäten als Quantum Computing as a Service (QCaaS) über die Cloud zur Verfügung zu stellen. Dadurch wird möglichst vielen Menschen der Zugang zu dieser Technologie ermöglicht. IBM stellt bereits diverse Quantencomputer und Simulatoren als QCaaS zur Verfügung und bietet mit IBM Q Experience [IBM] eine Plattform zur Programmierung von Quantenalgorithmen als Software as a Service (SaaS) an [MG11].

Die Programmierung von Quantenalgorithmen ist ein Spezialgebiet und es ist aufwendig sich in die verschiedenen Frameworks der Anbieter einzuarbeiten und das zugrunde liegende Paradigma zu verstehen. Außerdem sind die Frameworks in bestimmten Programmiersprachen geschrieben und können nicht ohne weiteres in bestehende Anwendungen integriert werden. Allerdings kann es sinnvoll sein eine klassische Anwendung durch Quantenalgorithmen zu unterstützen, um aufwendige Berechnungen, wie die Faktorisierung [Sho94], zu lösen. Um Entwicklern von klassischen Anwendungen eine programmiersprachenunabhängige Verwendung von Quantenalgorithmen zu ermöglichen, könnten Quantenalgorithmen als Quantum Algorithm as a Service (QAaaS) über die Cloud zur Verfügung gestellt werden. Dadurch könnten Entwickler die Schnittstellen der Services über die klassischen Anwendungen ansprechen und Quantenalgorithmen ausführen, sowie deren Ergebnisse erhalten, ohne etwas über die Implementierung und die Quantencomputer wissen zu müssen. Mit dieser Idee beschäftigt sich diese Arbeit.

Dafür werden zunächst verschiedene Quantencomputer-Paradigmen und Quanten-Frameworks untersucht. Aus den daraus gewonnenen Erkenntnissen wird eine Plattform konzipiert, indem Anforderungen spezifiziert werden, eine Architektur entworfen wird und empfohlene Praktiken für Quanten-Services formuliert werden. Mit der Plattform soll es möglich sein Quantenalgorithmen zu verwalten und Quanten-Services zu generieren. Ein Quanten-Service bietet eine Schnittstelle für die Ausführung eines Quantenalgorithmus an. Der Quanten-Service kann in der Cloud als

QAaaS angeboten werden. Über die Schnittstelle können beliebige klassische Anwendungen mit dem Quanten-Service kommunizieren, um Quantenberechnungen auszuführen. Anschließend wird das Konzept prototypisch umgesetzt und anhand eines Anwendungsbeispiels präsentiert.

Die Ausarbeitung ist in folgende Kapitel gegliedert:
Kapitel 2 vermittelt die Grundlagen für Quantencomputer-Paradigmen, Quantencomputing und Services. Kapitel 3 beschäftigt sich mit verwandten Arbeiten aus den Themenkomplexen Plattformen für Quantenalgorithmen, modulare Compiler und Generierung von Schnittstellen. Kapitel 4 befasst sich mit Quanten-Frameworks, die zur Erstellung von Quantenalgorithmen genutzt werden können und vergleicht sie miteinander. Kapitel 5 stellt die Konzeption der Plattform und der Quanten-Services vor. Kapitel 6 präsentiert die Implementierung des Konzepts und demonstriert die Funktionen der Plattform anhand eines Anwendungsbeispiels. Kapitel 7 beinhaltet die Zusammenfassung dieser Arbeit und einen Ausblick.

2 Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die für diese Ausarbeitung nützlich sind. Die drei behandelten Themenkomplexe sind Quantencomputer-Paradigmen, Quantencomputing und Services.

2.1 Quantencomputer-Paradigmen

Es gibt heute einen bestimmten Weg einen klassischen Computer aufzubauen. Die einzelnen Komponenten und ihre Bestandteile, sowie die Ausführung der Berechnung sind bei jedem klassischen Computer gleich. Sie folgen alle dem gleichen Paradigma. Quantencomputer befinden sich noch in der Anfangsphase ihrer Entwicklung und es gibt verschiedene Vorschläge Qubits zu implementieren und Berechnungen auszuführen, also verschiedene Paradigmen. Ursprünglich hatte DiVincenzo [DiV00] Kriterien für Quantencomputer herausgearbeitet, die einen Quantencomputer definieren. Mit der Entstehung neuer Paradigmen trafen die DiVincenzo-Kriterien nicht mehr allgemein zu, weshalb Pérez-Delgado und Kok [PK11] vier allgemeinere Kriterien formuliert haben, die für alle Paradigmen gelten sollen. (1) Ein Quantencomputer muss einen Quantenspeicher haben. (2) Es muss die Möglichkeit geben den Speicher zu verändern, sodass universelle Berechnungen möglich sind. (3) Der Speicher muss zurückgesetzt werden können. (4) Es muss eine Möglichkeit geben den Speicher auszulesen [PK11]. Ein Quantencomputer ist somit im Wesentlichen ein Speicher der beschrieben und gelesen werden kann, sodass wiederholt Berechnungen möglich sind.

Im folgenden Abschnitt werden vier verschiedene Paradigmen für Quantencomputer erläutert. Jedes Paradigma verfolgt einen anderen Weg die Qubits zu realisieren und Berechnungen auf ihnen auszuführen. Alle Paradigmen haben jedoch gemeinsam, dass der resultierende Quantencomputer die vier Kriterien von Pérez-Delgado und Kok [PK11] erfüllt und einen Speicher besitzt, um Berechnungen zu ermöglichen.

2.1.1 Gatterbasierte Quantencomputer

Ein Quantencomputer, der Berechnungen mithilfe von Gattern durchführt, wird als gatterbasierter Quantencomputer bezeichnet. Dieses Paradigma ähnelt stark der Berechnung auf klassischen Computern und ist daher leicht verständlich, weshalb die Erklärungen in Abschnitt 2.2 darauf beruhen. Ein Gatter ist eine unitäre Funktion, die auf einem oder mehreren Qubits ausgeführt wird. Eine Aneinanderreihung von Gattern wird als Schaltkreis bezeichnet [Hom15]. Einige Gatter für Quantencomputer gleichen in ihrer Funktion den Gattern für klassische Computer, wie es z. B. bei der Negation der Fall ist. Andere Gatter, wie das Hadamard-Gatter, sind speziell für Quantencomputer eingeführt. Einen deutlichen Unterschied gibt es bei der Umsetzung der Gatter. Während bei klassischen Gattern Transistoren zum Einsatz kommen, um die Logik zu realisieren, ist die Umsetzung

bei Quantencomputern komplizierter und vielfältiger. Monroe et al. [MMK+95] beschreiben eine Realisierung von zwei Qubits und eines Controlled-NOT (CNOT)-Gatters. Die Qubits werden mithilfe eines einzelnen Beryllium-Ions umgesetzt, indem zwei verschiedene binäre Eigenschaften des Ions betrachtet werden. Das CNOT-Gatter wird durch drei aufeinanderfolgende Laserstrahlen realisiert, die die Zustände der einzelnen Qubits verändern, sodass das Ziel-Qubit nur negiert wird, falls das Kontroll-Qubit den Wert 1 hat [MMK+95].

2.1.2 Adiabatische Quantencomputer

Bei adiabatischen Quantencomputern wird das gesamte System in einen vorbereiteten Grundzustand gebracht. Das System wird durch eine Hamilton-Funktion H_0 definiert und der Grundzustand ist der Zustand mit der geringsten Energie. Ziel ist es H_0 so zu verändern, dass es sich H_P des Problems annähert, aber den Grundzustand nicht verlässt. Farhi et al. [FGGS00] postulieren, dass die Definition von H_P einfach sei, es jedoch schwierig sei den zugehörigen Grundzustand herauszufinden. Das adiabatische Theorem besagt, dass ein System im Grundzustand bleibt, falls die Änderungen sehr langsam durchgeführt werden [DMV01]. Dies wird sich bei der adiabatischen Berechnung zunutze gemacht. Wird H_0 erfolgreich in H_P überführt, ohne dass der Grundzustand verlassen wird, kann er gemessen werden und entspricht der Lösung. Aharonov et al. [ADK+04] vergleichen den Grundzustand von H_P mit dem Endzustand einer gatterbasierten Berechnung, der ebenfalls der Lösung des Problems entspricht und kommen zu dem Ergebnis, dass gatterbasierte und adiabatische Berechnungen äquivalent sind [ADK+04] [YHW17].

2.1.3 Topologische Quantencomputer

Die Topologie ist ein Bereich in der Mathematik, der sich mit der elastischen Verformung geometrischer Objekte und ihren Eigenschaften beschäftigt. Ein bekannter Vergleich ist der zwischen einer Kaffeetasse und einem Donut, die sich durch kontinuierliche, weiche Verformungen ineinander überführen lassen. Es wird sich auf die globalen Eigenschaften eines Objekts konzentriert, sodass kleine lokale Änderungen keine Auswirkungen haben [SFN06]. Ein topologischer Quantencomputer nutzt diese Eigenschaften, wodurch er bereits auf Hardwareebene gegen lokale Fehler geschützt ist. Um einen Quantencomputer umzusetzen, werden Anionen benötigt. Anionen sind Quasiteilchen, die unter bestimmten Voraussetzungen in 2D-Systemen auftreten können [LP17]. Ein Qubit wird durch vier Anionen kodiert. Für jedes weitere Qubit werden zwei weitere Anionen benötigt. Die Quantengatter werden durch sogenanntes braiding realisiert, also die Verflechtungen, die durch das Vertauschen von Anionen entstehen. Im Beispiel von Lahtinen und Pachos [LP17] werden unitäre Funktionen auf bestimmte Anionen-Paare angewendet, um z. B. die Funktion des Hadamard-Gatters zu realisieren. In Abbildung 2.1 ist die Verflechtung zu sehen, die durch die Anwendung des Hadamard-Gatters auf das erste Qubit entsteht. Zunächst werden die Anionen an Stelle 1 und 2 vertauscht, anschließend 2 und 3 und als letztes erneut 1 und 2. Nach der Anwendung aller Gatter, werden die Anionen miteinander verschmolzen, um ein Messergebnis zu erhalten. Dafür werden die Anionen paarweise physikalisch zusammengeführt und es entsteht entweder ein Vakuum oder ein großes Teilchen. Der daraus resultierende Energieunterschied des Systems wird zur Ermittlung des Messergebnisses genutzt [LP17].

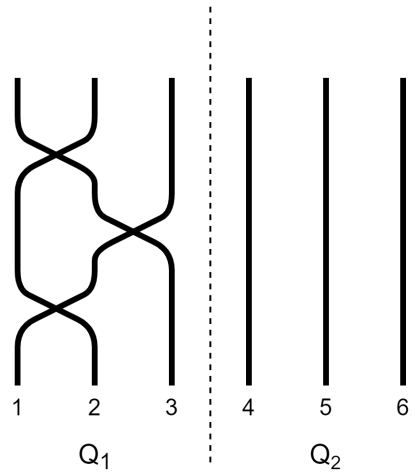


Abbildung 2.1: Die Verflechtung von drei Anionen, die durch die Anwendung des Hadarmard-Gatters auf das Qubit Q_1 entsteht. Auf Q_2 wird kein Gatter angewendet, sodass es zu keiner Verflechtung kommt.

2.1.4 Einweg-Quantencomputer

Raussendorf und Briegel [RB01] stellen mit dem Einweg-Quantencomputer ein weiteres Paradigma für Quantencomputer vor. Für die Berechnungen wird zu Beginn ein verschränkter Zustand auf einem Gitter aus Qubits erstellt, welcher als Cluster-Zustand bezeichnet wird. Auf diesem Zustand werden nur Operationen auf einzelnen Qubits ausgeführt, um Informationen auf das Cluster zu schreiben, zu verarbeiten und zu lesen. Es ist möglich einen Schaltkreis in das Gitter zu implementieren, indem einzelne Qubits vorab gemessen und damit unbrauchbar gemacht werden. Die verbliebenen Qubits ermöglichen auf der horizontalen des Gitters einen Informationsfluss und auf der vertikalen die Umsetzung von 2-Qubit-Gattern, wie z. B. dem CNOT-Gatter, das sie in ihrer Arbeit vorstellen. Während der Berechnung werden die Qubits von links nach rechts in einer bestimmten Basis gemessen, die vom Ergebnis der vorherigen Messung abhängt. Die Messung der letzten Qubits, in der entsprechenden Basis, liefert das Ergebnis [RB01]. Walther et al. [WRR+05] haben in ihrem Experiment einen echten Cluster-Zustand erstellt. Dazu haben sie aus der Polarisation von vier Photonen vier Qubits erzeugt und sie miteinander verschränkt. Auf dem erzeugten Cluster-Zustand haben sie den Grover-Algorithmus implementiert und gezeigt, dass Einweg-Quantencomputer derartige Probleme lösen können [WRR+05].

2.2 Quantencomputing

In diesem Abschnitt werden die Grundlagen für Berechnungen auf einem gatterbasierten Quantencomputer erläutert.

2.2.1 Quantenbits

Bei klassischen Computern sind klassische Bits (Cbits) die kleinste Informationseinheit. Sie können die Werte 0 oder 1 annehmen. Diese Werte lassen sich als Vektoren in der Dirac-Notation [Dir39] als $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ bzw. $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$ schreiben. In Quantencomputern verkörpern Qubits die kleinste Informationseinheit. Ein Qubit ist ein Zweizustand-Quantensystem, also ein System, das quantenmechanische Eigenschaften besitzt und zwei messbare gegensätzliche Zustände einnehmen kann. Qubits können die Werte 0, 1 oder eine Linearkombination aus den beiden Werten, die sogenannte Superposition, annehmen. Ein Qubit hat die Form $|\psi\rangle = \alpha * |0\rangle + \beta * |1\rangle = \begin{pmatrix}\alpha\\\beta\end{pmatrix}$ mit $|\alpha|^2 + |\beta|^2 = 1$. α und β werden als Amplituden bezeichnet. Wird ein Qubit gemessen, geht es mit Wahrscheinlichkeit $|\alpha|^2$ in den Zustand $|0\rangle$ und mit Wahrscheinlichkeit $|\beta|^2$ in den Zustand $|1\rangle$ über [Hom15]. Beispiele für Qubits in Superposition sind

$$|+\rangle = \frac{1}{\sqrt{2}} * (|0\rangle + |1\rangle) = \begin{pmatrix}\frac{1}{\sqrt{2}}\\\frac{1}{\sqrt{2}}\end{pmatrix}, |-\rangle = \frac{1}{\sqrt{2}} * (|0\rangle - |1\rangle) = \begin{pmatrix}\frac{1}{\sqrt{2}}\\-\frac{1}{\sqrt{2}}\end{pmatrix} \text{ und } \begin{pmatrix}\frac{1}{2}\\\frac{\sqrt{3}}{2}\end{pmatrix} \tag{2.1}$$

Ein Qubit kann beispielsweise durch ein von Umwelteinflüssen abgeschottetes Photon erzeugt werden. Die Abschottung ist essenziell, da es ansonsten zur Dekohärenz kommt und die Superposition zerstört wird [Hom15]. Photonen haben eine Polarisation. Nun können die Photonen mit vertikaler Polarisation als Qubit in Zustand $|0\rangle$ und die Photonen mit horizontaler Polarisation als Qubit in Zustand $|1\rangle$ definiert werden. Photonen, die anders polarisiert sind, sind Qubits in Superposition. Mithilfe eines Polarisationsstrahlteilers können die Qubits gemessen werden. Dabei wird die Superposition zerstört und die Werte 0 oder 1 gemessen. Der Aufbau ist schematisch in Abbildung 2.2 dargestellt. Vertikal polarisierte Photonen, also Qubits in Zustand $|0\rangle$, werden durchgelassen und horizontal polarisierte Photonen abgelenkt. In Superposition befindliche Qubits werden mit den jeweiligen Wahrscheinlichkeiten $|\alpha|^2$ und $|\beta|^2$ durchgelassen oder abgelenkt. Detektoren registrieren das Auftreffen der Photonen und messen die Qubits [Hom15].

2.2.2 Quantenberechnungen

Eine Berechnung ist eine Folge von Rechenschritten, die das System in neue Zustände überführen. In jedem Rechenschritt werden ein oder mehrere Cbits respektive Qubits betrachtet und durch das Anwenden von Gattern verändert oder beibehalten, um dem gewünschten Endzustand näher zu kommen. Bei klassischen Computern können die Cbits beliebig auf 0 oder 1 gesetzt werden bzw. unverändert bleiben. Ein legaler Rechenschritt in einem klassischen System ist die Formel 2.2. Das Gatter K_1 weist einem beliebigen Cbit stets den Wert 1 zu.

$$K_1 |0\rangle = \begin{pmatrix}0 & 0\\1 & 1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix} \text{ und } K_1 |1\rangle = \begin{pmatrix}0 & 0\\1 & 1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0\\1\end{pmatrix} \tag{2.2}$$

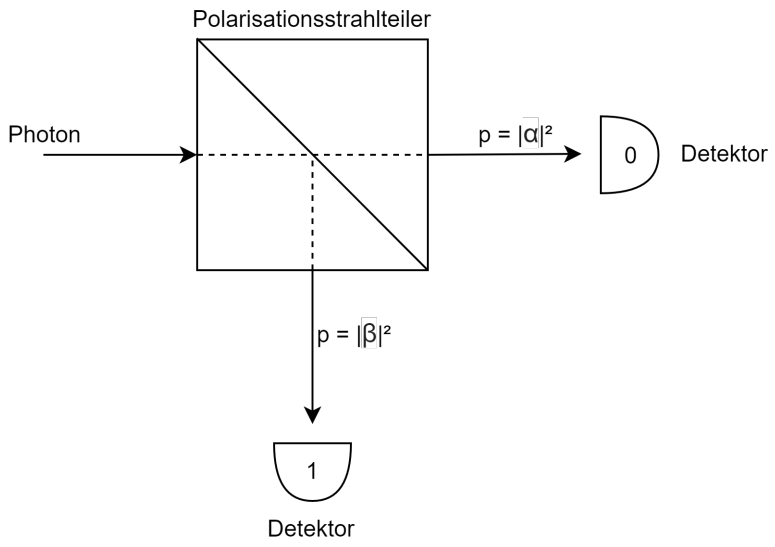


Abbildung 2.2: Ein Photon trifft auf einen Polarisationsstrahlteiler und wird je nach seiner Polarisation, abhängig von α und β, durchgelassen oder abgelenkt. Detektoren registrieren die auftreffenden Photonen, um die Werte 0 und 1 unterscheiden zu können.

Im Gegensatz dazu müssen bei Quantencomputern die Rechenschritte und damit die gesamte Berechnung reversibel sein, d. h. dass es möglich sein muss das Ergebnis einer Berechnung eindeutig in den Ausgangszustand zurückzuführen [Hom15]. Die Matrizen aus Formel 2.2 besitzen keine Inverse und aus den Ergebnissen der Berechnungen ist nicht eindeutig ersichtlich, welchen Wert das jeweilige Qubit vor der Berechnung hatte. Damit eine Berechnung reversibel ist, muss die Matrix des Gatters unitär sein. Eine Matrix M heißt unitär, wenn die transponierte Matrix M^T zu M invers ist, also $M^T = M^{-1}$ gilt. Ein Beispiel hierfür ist das Hadamard-Gatter H aus Formel 2.3. Die Anwendung des Hadamard-Gatters bringt ein Qubit in Superposition [Hom15].

$$H\,|0\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \text{ und } H\,|1\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{2.3}$$

Die Ergebnisse aus Formel 2.3 unterscheiden sich durch ein negatives Vorzeichen, da die Anwendung des Hadamard-Gatters $|0\rangle$ und $|1\rangle$ in verschiedene Zustände überführt, um die Reversibilität zu gewährleisten. Auf eine nachfolgende Messung hat das Vorzeichen keine Auswirkungen, da die Wahrscheinlichkeiten in beiden Fällen gleich sind.

Berechnungen können nicht nur auf einem einzelnen Qubit, sondern auch auf mehreren Qubits ausgeführt werden. Mehrere Qubits bilden zusammen ein Quantenregister R, das beispielsweise aus zwei Qubits im Zustand $|1\rangle$, mit $R = |11\rangle$, besteht. Der Zustandsvektor für das Quantenregister R wird aus dem Tensorprodukt der Zustandsvektoren aller Qubits berechnet [Hom15], wie es in Formel 2.4 dargestellt ist.

$$R = |11\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ 1 \begin{pmatrix} 0 \\ 1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.4}$$

Ein Beispiel für ein Gatter, das auf einem Quantenregister arbeitet, ist das CNOT-Gatter. Es entspricht in der Funktionsweise dem klassischen XOR-Gatter. Die Eingabe für das CNOT-Gatter besteht aus zwei Qubits, dem Kontrollbit und dem Zielbit. Ist das Kontrollbit im Zustand $|1\rangle$, wird beim Passieren des CNOT-Gatters das Zielbit negiert, ansonsten bleibt das Zielbit unverändert [Hom15]. Die Anwendung des CNOT-Gatters auf das Register R ist in der Formel 2.5 dargestellt. Durch die Anwendung des CNOT-Gatters wird das zweite Qubit negiert, wodurch das Register am Ende der Berechnung im Zustand $|10\rangle$ ist.

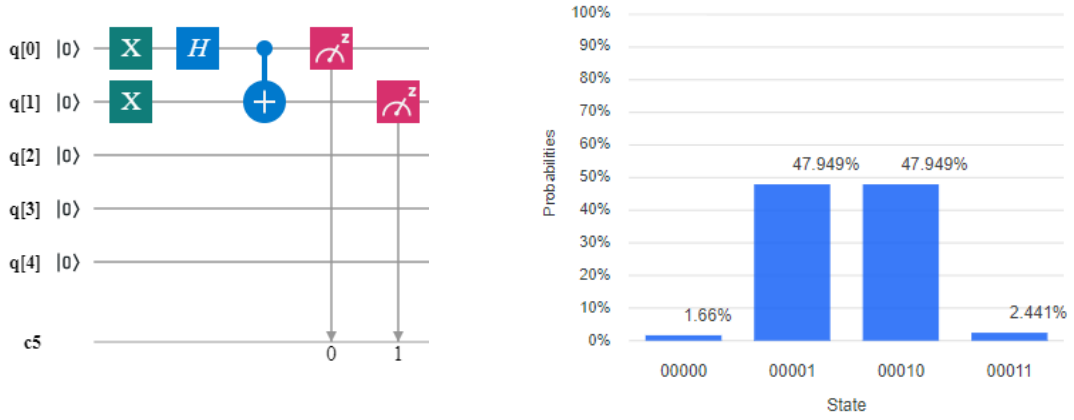
$$CNOT\,|11\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ 1 \begin{pmatrix} 1 \\ 0 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |10\rangle \tag{2.5}$$

In Kombination können Hadamard- und CNOT-Gatter dazu verwendet werden Qubits miteinander zu verschränken. Zwei verschränkte Qubits haben die Eigenschaft, dass sie nicht mehr getrennt voneinander betrachtet werden können. Ein Rechenbeispiel ist in Formel 2.6 gegeben. Um die beiden Qubits aus Register R miteinander zu verschränken, wird zunächst das Hadamard-Gatter auf das erste Qubit angewendet. Anschließend wird das CNOT-Gatter auf das Register angewendet [Hom15].

$$|11\rangle \xrightarrow{H*Q_1} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} \xrightarrow{CNOT} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} \tag{2.6}$$

Nach der Definition von Homeister [Hom15] gilt ein Register als verschränkt, wenn der Zustandsvektor des Registers nicht in die Zustandsvektoren der einzelnen Qubits zerlegt werden kann. Diese Zerlegung ist bei dem resultierenden Zustandsvektor aus dem Beispiel in Formel 2.6 nicht möglich. Die beiden Qubits sind nun miteinander verschränkt. Der Zustandsvektor kann als $\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$ formuliert werden. Durch den Term wird deutlich, dass die Qubits voneinander abhängen, da jeweils mit Wahrscheinlichkeit $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$ entweder $|01\rangle$ oder $|10\rangle$ das Messergebnis ist. Angenommen das erste Qubit wird gemessen und ergibt den Wert 0, dann kann eine Messung des zweiten Qubit nur den Wert 1 ergeben und anders herum. In Abbildung 2.3a ist ein Schaltkreis zu sehen, der mithilfe von IBM Q Experience [IBM] erstellt wurde. Der Schaltkreis entspricht der Berechnung aus Formel 2.6 und wurde auf dem Quantencomputer Vigo von IBM (siehe Tabelle 4.2) ausgeführt. Das Ergebnis der Verschränkung ist in Abbildung 2.3b zu sehen. Da der Quantencomputer fünf Qubits besitzt, werden diese beim Ergebnis in der Abbildung 2.3b angezeigt. In ca. 96% der Fälle ergab eine Messung die Ergebnisse $|01\rangle$ und $|10\rangle$. Ca. 4% der Fälle sind ein unerwartetes Ergebnis, was z. B. durch Umwelteinflüsse oder die Ungenauigkeit der Gatter auftreten kann. Durch die Reversibilität im Quantencomputing kann die Verschränkung rückgängig gemacht werden, indem das CNOT-Gatter erneut angewendet wird.

Eine weitere Besonderheit von Qubits ist es, dass die Informationen eines Qubits nicht kopiert werden können. Das No-Cloning-Theorem besagt, dass es keine unitäre Operation gibt, um ein beliebiges Qubit in ein anderes zu kopieren [Hom15]. Bei einem klassischen Computer stellt diese Aufgabe kein Problem dar, da mithilfe des CNOT-Gatters die Information des Kontrollbits in das Zielbit kopiert werden kann, wenn das Zielbit mit $|0\rangle$ initialisiert ist. So ergibt $CNOT\,|00\rangle = |00\rangle$ und $CNOT\,|10\rangle = |11\rangle$. Das CNOT-Gatter ist unitär und wäre somit eine Möglichkeit Qubits zu



(a) Schaltkreis

(b) Ergebnis

Abbildung 2.3: Demonstration einer Verschränkung auf einem Quantencomputer von IBM.

kopieren [Hom15]. Mit Qubits ist das allerdings nur möglich, wenn sich die Zustände ebenfalls auf $|0\rangle$ und $|1\rangle$ beschränken. Befinden sich die Qubits in Superposition, führt die Anwendung des CNOT-Gatters zur Verschränkung und das Qubit wird nicht kopiert. Dementsprechend ist das CNOT-Gatter, genau wie jede andere unitäre Operation, nicht für das Kopieren von Qubits geeignet, wie aus den Ausführungen von Homeister [Hom15] hervorgeht.

2.2.3 Quantenalgorithmen

In diesem Abschnitt werden zwei bekannte Quantenalgorithmen vorgestellt, die beispielhaft darstellen, wie eine Berechnung eines konkreten Problems auf einem Quantencomputer umgesetzt werden kann. Außerdem zeigen die Beispiele, dass Quantencomputer bestimmte Probleme effizienter lösen können, als klassische Systeme.

Grover-Algorithmus

Der Grover-Algorithmus wurde 1996 von Grover [Gro96] als Suchalgorithmus für Datenbanken vorgestellt. Er ist ein gutes Beispiel für einen Quantenalgorithmus, der eine Suche schneller durchführen kann, als die klassischen Varianten. Ein klassischer Suchalgorithmus braucht auf N Datenbankelementen im schlimmsten Fall $O(N)$, um das gesuchte Element zu finden. Der Algorithmus von Grover beendet eine Suche auf einem Quantencomputer spätestens nach $O(\sqrt{N})$ und hat einen Speicherbedarf von $O(\log N)$.

Um den Grover-Algorithmus durchzuführen, wird zunächst das gesamte System $|s\rangle$ in Superposition gebracht, indem auf alle $|x\rangle$ das Hadamard-Gatter angewendet wird. Um die Suche durchführen zu können benötigt der Algorithmus ein Orakel U_f als Eingabe, das Informationen zum gesuchten Element $\hat{x}$ und der Datenstruktur enthält. f ist eine Funktion mit

$$f(x) = \begin{cases} 1, & \text{falls } x = \hat{x} \\ 0, & \text{falls } x \neq \hat{x} \end{cases} \tag{2.7}$$

Außerdem wird ein Hilfs-Qubit $|y\rangle$ benötigt, um die Reversibilität von f sicherzustellen. $|y\rangle$ wird als $|1\rangle$ initialisiert und durch das Hadamard-Gatter in den Zustand $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ gebracht. Das Orakel U_f wirkt auf $|x\rangle\,|y\rangle$ wie folgt [Hom15]:

$$|x\rangle\,\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \xrightarrow{U_f} |x\rangle\,\frac{1}{\sqrt{2}}(|f(x)\rangle - |1 \oplus f(x)\rangle) = (-1)^{f(x)}\,|x\rangle\,\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.8}$$

Für $x = \hat{x}$ wertet $f(x)$ zu 1 aus, wodurch das Vorzeichen von $\hat{x}$ negiert wird. In den anderen Fällen ändert sich das Vorzeichen nicht. Eine Messung im jetzigen Zustand würde alle Elemente mit gleicher Wahrscheinlichkeit ausgeben. Um die Wahrscheinlichkeit des gesuchten Elements zu erhören, wird eine Amplitudenverstärkung durchgeführt, indem jede Amplitude am Mittelwert aller Amplituden gespiegelt wird. Danach ist die Amplitude für $\hat{x}$ höher. In Abbildung 2.4 wird dieser Effekt an einem Beispiel deutlich gemacht. Zu Beginn haben alle vier Elemente eine Amplitude von $\frac{1}{2}$. Nach der Anwendung von U_f hat das gesuchte Element die Amplitude $-\frac{1}{2}$, die anderen weiterhin die Amplitude $\frac{1}{2}$. Wird nun am Mittelwert der Amplituden, also $\frac{1}{4} * (3 * \frac{1}{2} - \frac{1}{2}) = \frac{1}{4}$, gespiegelt, erhält $\hat{x}$ die Amplitude 1 und die anderen Elemente die Amplitude 0. Da die Amplitude von $\hat{x}$ groß genug ist, kann gemessen werden. Ansonsten würde die Amplitudenverstärkung wiederholt werden, bis die Amplitude von $\hat{x}$ groß genug ist [Hom15]. Sollte die Amplitudenverstärkung zu oft durchgeführt werden, kann sie den gegenteiligen Effekt haben und die Amplitude von $\hat{x}$ wieder verringern, wodurch die Berechnung unbrauchbar wird. Für n Qubits in x sollte sie $\lceil \frac{\pi 2^{\frac{n}{2}}}{4} \rceil$ wiederholt werden [CEP+18].

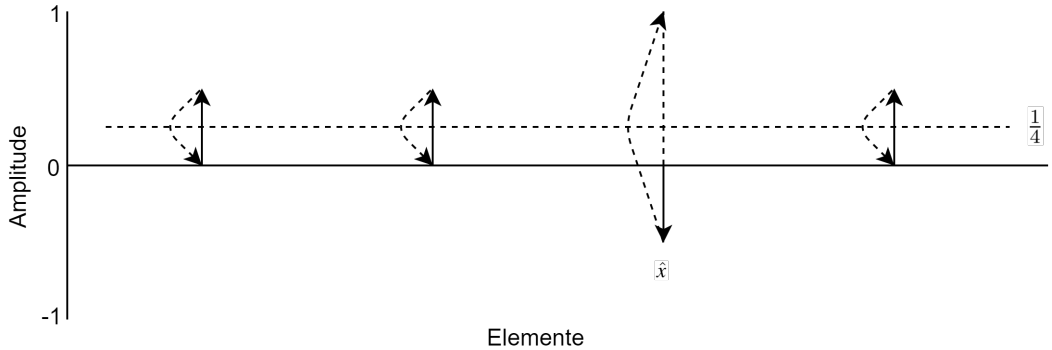


Abbildung 2.4: Spiegelung der Amplituden am Mittelwert aller Amplituden am Beispiel von vier Elementen. Durch die erhöhte Amplitude von $\hat{x}$, wird die Wahrscheinlichkeit für dieses Element erhöht.

Shors Algorithmus

Der Shor-Algorithmus wurde 1994 vorgestellt und ist nach dem Erfinder Peter Shor benannt [Sho94]. Der Algorithmus kann zur Lösung des Faktorisierungsproblems genutzt werden. Ziel ist es für eine Zahl N zwei ganze Zahlen N_1, N_2 mit $1 < N_1, N_2 < N$ zu finden, sodass $N = N_1 * N_2$. Dieses Problem ist für klassische Computer sehr schwierig zu lösen, sodass das Problem für Zahlen, die aus mehr als 1000 Cbits bestehen, als unlösbar gilt [CEP+18]. Diese Eigenschaft bildet die Grundlage für das RSA-Verfahren, welches ein sehr verbreitetes Verschlüsselungsverfahren ist. Theoretisch könnte Shors Algorithmus das Faktorisierungsproblem für weitaus größere Zahlen

auf einem Quantencomputer lösen, was das RSA-Verfahren unbrauchbar macht. Jedoch ist die größte faktorisierte Zahl auf einem Quantencomputer die Zahl 21 [CEP+18], weshalb das RSA-Verfahren weiterhin sicher bleibt. Coles et al. [CEP+18] beschreiben zwei Möglichkeiten, um Shors Algorithmus umzusetzen und auf einem Quantencomputer auszuführen.

2.3 Services

Ein Service ist eine in sich geschlossene Komponente, die modular aufgebaut ist und unabhängig von anderen Komponenten bereitgestellt werden kann. Ein Service bietet eine Funktionalität nach außen an, die von anderen Komponenten über das Netzwerk in Anspruch genommen werden kann. Die Funktionalität ist gekapselt und wird über ein Interface bereitgestellt, sodass die konkrete Implementierung nach außen hin unsichtbar ist. Dieses Prinzip wird als Information Hiding bezeichnet. Das Interface muss plattform- und programmiersprachenunabhängig sein, sodass der Service universell verwendbar ist [BKM07] [Mel10]. In Bezug auf die Service-orientierte Architektur (SOA) führen Bianco et al. [BKM07] zwei weitere Kriterien an. Ein Service sollte durch einen speziellen Verzeichnisservice auffindbar sein und er sollte dynamisch zur Laufzeit gebunden werden, jedoch nicht zum Build-Zeitpunkt eines Systems. Diese beiden Punkte werden nach ihren Angaben in der Realität selten oder in abgeschwächter Form umgesetzt [BKM07].

RESTful Webservices

REST steht für Representational State Transfer und definiert eine Menge an Regeln, um Webservices zu erstellen. Rodriguez [Rod08] postuliert, dass REST sich als Designmodell für Webservices durchgesetzt hat. REST wurde zum ersten Mal 2000 von Fielding [Fie00] in seiner Dissertation vorgestellt. Ein Webservice sollte sich an die Prinzipien des Client-Server-Modells halten, also die Benutzeroberfläche von der Funktionalität trennen. Des Weiteren soll die Kommunikation zwischen Client und Server zustandslos verlaufen, d. h. eine Anfrage des Clients muss alle relevanten Informationen enthalten, damit der Server die Anfrage verarbeiten kann. Es werden keine Session-Informationen auf dem Server gespeichert. Das bringt Vorteile in den Bereichen Sichtbarkeit, Zuverlässigkeit und Skalierbarkeit [Fie00]. Um die Geschwindigkeit von Anfragen und die Netzwerkauslastung zu verbessern, sollen Ressourcen in einem Cache gespeichert werden können. Das hat den Vorteil, dass die Ressourcen bei zukünftigen Anfragen nicht erneut übertragen werden müssen, sondern aus dem Cache des Clients geladen werden können. Allerdings kann dieses Vorgehen zu Nachteilen in der Zuverlässigkeit führen, da die Version des Servers aktueller sein kann, als die Version im Cache. Der Hauptunterschied zwischen REST und anderen Architekturstilen ist die einheitliche Schnittstelle. Vier Eigenschaften müssen erfüllt sein, um die einheitliche Schnittstelle umzusetzen. Zunächst müssen alle Ressourcen mithilfe eines Ressource Identifiers (URI) eindeutig identifiziert werden können. Außerdem sollen Aktionen auf Ressourcen nur über Repräsentationen stattfinden, die aus den Daten und Metadaten bestehen. Dadurch können die Ressourcen über die Kontrolldaten einer Anfrage verändert werden und eine Antwort kann, je nach angefragtem Media-Type, den gewünschten Dateityp ausgeben. Des Weiteren sollen Nachrichten selbstbeschreibend sein und Informationen zum Empfänger und der gewünschten Aktion beinhalten. In HTTP werden Standardmethoden, wie z. B. GET und POST, verwendet, um die gewünschten Aktionen in der Nachricht zu übermitteln. Zuletzt wird die Umsetzung von hypermedia as the engine of application state

(HATEOAS) gefordert. Das bedeutet, dass die Navigation einer HTML-Seite über Links geschieht. Je nachdem welcher Link ausgewählt wird, befindet sich der Client in einem anderen Zustand und erhält andere Interaktionsmöglichkeiten. Dadurch kann dem Nutzer zunächst eine kleine Menge an Links präsentiert werden, um die Komplexität des Service zu verringern [VB15]. Ein System sollte zudem mehrschichtig aufgebaut sein. Für den Client ist nur die Schnittstelle des äußersten Layer sichtbar. Tieferliegende Layer, die Operationen im Hintergrund ausführen, sind nicht sichtbar, um die Komplexität nach außen hin zu verschleiern. Die letzte Regel ist optional und erlaubt es die Funktionalität auf einem Client zu erweitern, indem Code heruntergeladen und ausgeführt wird. Ein Beispiel dafür ist das Übertragen und Ausführen von Javascript-Code beim Besuch einer Webseite [Fie00].

3 Verwandte Arbeiten

In diesem Kapitel werden die Arbeiten vorgestellt, die zu dieser Arbeit ähnliche Problemstellungen und Konzepte beschreiben.

3.1 Plattform für Quantenalgorithmen

Der Quanten-Simulator quantum-circuit [Korb] wurde für Javascript entwickelt. Mit dem Simulator können Quantenalgorithmen auf über 20 Qubits erstellt und simuliert werden. Außerdem ist es möglich die Algorithmen in die Sprachen anderer Frameworks zu exportieren. Zu den unterstützten Sprachen zählen unter anderem Qiskit, Cirq und pyQuil. Ende 2018 hat Petar Korponaić eine Benutzeroberfläche für quantum-circuit entwickelt. Dadurch ist das Quantum Programming Studio (QPS) [Kora] entstanden. QPS ist eine Plattform, auf der die Nutzer Quantenalgorithmen verwalten, teilen und ausführen können. Es ist möglich einen neuen Algorithmus in einem grafischen Editor per Drag & Drop zusammenzusetzen oder ihn programmatisch in Open Quantum Assembly Language (OpenQASM) [CBSG17] zu erstellen. Außerdem gibt es die Möglichkeit die veröffentlichten Algorithmen anderer Nutzer zu betrachten und in den eigenen Arbeitsbereich zu klonen. Des Weiteren ist es möglich die Algorithmen zu simulieren oder auf einem Quantencomputer von Rigetti oder IBM auszuführen. Im Designer ist es ebenfalls möglich die geöffneten Algorithmen in verschiedenen Sprachen oder als Grafiken zu exportieren [Kora].

In QPS können nur Algorithmen importiert werden, welche mit OpenQASM programmiert sind, indem der Code manuell in den Editor kopiert wird. Der Import von Algorithmen, die in anderen Sprachen bzw. Frameworks programmiert sind, ist nicht möglich. Außerdem ist es in QPS nicht möglich die Algorithmen zu parametrisieren, sodass der Code nicht durch Input-Parameter beeinflusst werden kann. Die Algorithmen werden auf der Plattform ausgeführt.
In dieser Arbeit wird ein Konzept für eine Plattform entwickelt, das Quantenalgorithmen, unabhängig vom verwendeten Framework, verwalten kann. Aus den Quantenalgorithmen können Quanten-Services erstellt werden, die unabhängig von der Plattform betrieben werden und über Parameter verfügen, um die Eingabe zu variieren.

3.2 Modulare Compiler

Es gibt viele Hersteller von Quantencomputern und jeder Hersteller stellt sein eigenes Framework zur Verfügung, um die Backends ansprechen zu können. Das führt dazu, das ein Algorithmus für jedes Backend neu programmiert werden muss. Dieses Problem versuchen Provider-agnostische Frameworks, wie ProjectQ [SHT18] zu umgehen. Ein Lösungsansatz, um verschiedene Backends zu unterstützen, sind modulare Compiler. Sie erlauben den Kompiliervorgang individuell festzulegen,

sodass der Code für verschiedene Backends kompiliert werden kann. Wie in Abschnitt 4.7 beschrieben wird, verwendet ProjectQ diesen Lösungsansatz. Zum Zeitpunkt dieser Arbeit unterstützt ProjectQ nur die Backends von IBM Q Experience [IBM].
Es ist sehr aufwendig einen Compiler zu entwickeln. Ekman und Hedin [EH07] haben das JastAdd-System entwickelt, um Entwickler bei der Erstellung von modularen Compilern zu unterstützen. Um die Vorteile des Tools zu demonstrieren haben Ekman und Hedin [EH07] den Java-1.4-Compiler und modularen Erweiterungen mit JastAdd implementiert. Der Code, der für die Compiler-Spezifikation in JastAdd benötigt wurde, war über ein Drittel kleiner als der Code des handgeschriebenen Compilers [EH07].

Einen Compiler zu entwickeln bleibt trotz Tool-Unterstützung eine aufwändige Aufgabe. In dieser Arbeit wird auf die Entwicklung einzelner Compiler für jedes Framework verzichtet. Stattdessen gibt es die Möglichkeit verschiedene Implementierungen zu gruppieren, sodass ein Service trotzdem mit verschiedenen Backends funktioniert. Dieses Thema wird in Kapitel 5 weiter ausgeführt.

3.3 Generierung von Schnittstellen

Programmierschnittstellen, deutsch für Application Programming Interfaces (APIs), sind heute in den meisten Anwendungen zu finden. APIs stellen eine definierte Menge an Funktionen bereit, die von anderen Anwendungen aufgerufen werden können. Die Implementierungsdetails der Funktionen bleiben der aufrufenden Anwendung unbekannt. APIs können für eine bestimmte Programmiersprache oder programmiersprachenunabhängig bereitgestellt werden. Eine Bibliothek, die in eine Anwendung eingebunden werden kann, ist gewöhnlich für eine bestimmte Programmiersprache entwickelt. Ein Beispiel für programmiersprachenunabhängig APIs sind Webservices, die eine REST API bereitstellen. Die API des Webservice kann über HTTP von jeder Anwendung, unabhängig von der Programmiersprache, aufgerufen werden, da alle verbreiteten Programmiersprachen Möglichkeiten anbieten HTTP-Anfragen zu versenden. Dadurch bieten sie eine gute Möglichkeit, um verschiedene Anwendungen in verschiedenen Programmiersprachen zu integrieren. Allerdings bieten nicht alle Artefakte, wie Anwendungen, Skripte und ausführbaren Dateien, eine API an oder haben sämtliche Funktionalität durch eine API verwendbar gemacht. Für diesen Fall haben Wettinger et al. [WBL+15] das Framework Any2Api entwickelt, das automatisiert APIs für verschiedene Artefakte erstellt. Für die automatisierte Erstellung einer API wird neben dem Artefakt eine Datei mit Metadaten benötigt, die unter anderem die Eingabeparameter der API definiert. Anschließend kann die API in sieben Schritten generiert werden [WBL+15].

Wettinger et al. [WBL+15] verfolgen einen generischen Ansatz, um möglichst viele Artefakte zu unterstützen. In dieser Arbeit geht es um die Generierung von Schnittstellen für Quantenalgorithmen. Ein Quantenalgorithmus ist, in Bezug auf die untersuchten Frameworks, ein Python-Skript mit einer Methode, die den Algorithmus erstellt und ausführt. Um eine Schnittstelle für einen Quantenalgorithmus zu generieren, wird er in einen Service eingebettet, der die definierte Funktionalität über eine HTTP-Schnittstelle zur Verfügung stellt, was in Kapitel 5 näher beschrieben wird.

4 Quanten-Frameworks

In diesem Kapitel werden die untersuchten Quanten-Frameworks vorgestellt. Die ausgewählten Frameworks erfüllen die Bedingungen freie Verfügbarkeit und ausführliche Dokumentation. Weitere Frameworks und andere Tools sind in den Listen von Quantum Computing Report [Finb] und der Quantum Open Source Foundation [Fou] aufgeführt.

Die grundlegende Architektur und der daraus resultierende Ablauf ist bei den meisten Frameworks sehr ähnlich und in Abbildung 4.1 dargestellt. Im ersten Schritt wird in der jeweiligen Programmiersprache bzw. domänenspezifischen Sprache (DSL) ein Programm geschrieben und ggf. die Konfiguration, für z. B. Compiler und Backend, festgelegt. Der Quellcode wird an den Compiler übergeben, der alle Kompilierschritte ausführt und den entstandenen Maschinencode an das definierte Backend sendet. Das Backend kann z. B. ein lokaler Simulator, ein externer Quantencomputer oder eine Visualisierungs-Engine sein, die den Maschinencode verarbeitet und das Ergebnis an das Programm zurückgibt. Die konkrete Umsetzung ist in jedem Framework unterschiedlich.

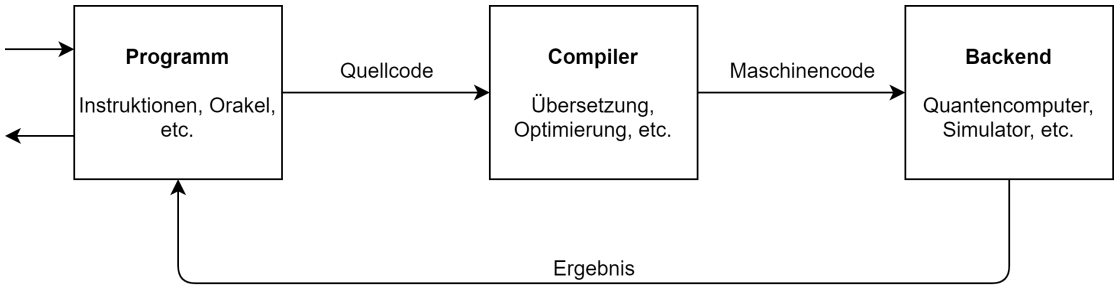


Abbildung 4.1: Generische Architektur der untersuchten Frameworks.

In Tabelle 4.1 werden die Frameworks anhand weiterer Merkmale verglichen und eine Übersicht gegeben. Zu den betrachteten Merkmalen zählen die Entwickler des Frameworks, ob sie Provider-agnostisch sind, die zu verwendende Programmiersprache, das zugrundeliegende Quantencomputer-Paradigma, die kleinste Informationseinheit, ob es Simulatoren und Quantencomputer gibt, sowie die Lizenz.
Nicht Provider-agnostische, also Provider-spezifische, Frameworks werden von den Herstellern der Quantencomputer zur Verfügung gestellt. Mit ihnen ist es möglich Instruktionen an die Quantencomputer des jeweiligen Herstellers zu senden, jedoch nicht an andere Backends, wodurch sie weniger flexibel sind. Dem gegenüber stehen Provider-agnostische Frameworks, die von Drittanbietern entwickelt werden. Sie verfolgen das Ziel verschiedene Backends von verschiedenen Herstellern verwenden zu können und eine hohe Flexibilität zu gewährleisten.
Es ist ersichtlich, dass die meisten Frameworks auf Python basieren. Die Ausnahmen sind Strawberry Fields mit der Python-ähnlichen Sprache Blackbird und Microsofts Quantum Development Kit (QDK) mit Q#. Blackbird ist in Strawberry Fields integriert, sodass die Programmierung in einem Python-Skript stattfindet. Q# bietet ein Python-Modul an, mit dem der Import und die Ausführung

von Q#-Operationen in einem Python-Skript möglich ist.
Weitere Unterschiede sind in den Quantencomputer-Paradigmen zu sehen. IBM, Google, Forest und Xanadu setzen das gatterbasierte Paradigma um. Microsoft verwendet topologische Qubits und versucht einen topologischen Quantencomputer zu bauen, der allerdings noch nicht existiert [LaR19]. D-Wave konzentriert sich auf adiabatische Quantencomputer, die sie bereits kommerziell anbieten. Die Provider-agnostischen Frameworks verwenden in ihrer Syntax Gatter zur Programmierung der Algorithmen, weshalb sie als gatterbasiert kategorisiert sind. Eine interne Übersetzung ermöglicht die Ausführung auf topologischen oder adiabatischen Backends. Ein Einweg-Quantencomputer wird von keinem der betrachteten Provider angeboten und von keinem untersuchten Framework unterstützt.
Im Vergleich zu allen anderen Frameworks verwendet Xanadu keine Qubits als kleinste Informationseinheit, sondern Qumodes und es kommen andere Gatter zum Einsatz. Umgesetzt werden die Qumodes mithilfe von Photonen [KIQ+19].
Simulatoren werden von allen Frameworks angeboten. Unterschiede treten bei der Verfügbarkeit von Quantencomputern auf. In Tabelle 4.1 werden die verfügbaren Quantencomputer durch Zahlen angegeben, die als Schlüssel für Tabelle 4.2 dienen. Cirq, Strawberry Fields und QDK bieten keinen Quantencomputer als Backend an. PennyLane bietet selbst keinen Quantencomputer als Backend zur Auswahl an. Durch die Verwendung der Plugins für die anderen Frameworks, ist ein Quantencomputer als Backend verwendbar.
Bei dem Merkmal Lizenz gibt es wenige Unterschiede. Die meisten Frameworks sind unter der Apache-2.0-Lizenz verfügbar. Die Ausnahmen sind das QDK, das durch die MIT-Lizenz lizenziert ist und XACC, dass als Eclipse-Projekt unter der EPL veröffentlicht ist.

In Tabelle 4.2 sind konkrete Quantencomputer zu sehen, die über das jeweilige Framework aus Tabelle 4.1 verwendet werden können. IBM stellt den Nutzern der IBM Q Experience vier verschiedene Quantencomputer zur Verfügung [IBM]. Drei der Quantencomputer besitzen 5 und einer 14 Qubits [IBM]. Rigetti bietet einen Quantencomputer mit 16 Qubits an [Rigb]. Die gatterbasierten Quantencomputer von IBM und Rigetti verwenden Supraleiter, um ihre Qubits umzusetzen. D-Wave verwendet ebenfalls Supraleiter, stellt allerdings adiabatische Quantencomputer her und bietet einen Quantencomputer mit 2048 Qubits an [D-Wc]. Die Zahlenwerte zu Qubit Verknüpfung, T1, T2 und Auslesegenauigkeit geben Auskunft über die Qubit-Qualität der einzelnen Quantencomputer. Die Qubit Verknüpfung gibt an, wie viele Qubits im Durchschnitt miteinander verbunden sind, um ein CNOT-Gatter aufzubauen. Eine höhere Verknüpfung resultiert in mehr Möglichkeiten eine Berechnung auf diesem Quantencomputer durchführen zu können [Fina]. Es ist auffällig, dass es bei den 5-Qubit-Quantencomputern von IBM Unterschiede gibt. Dies ist auf die jeweilige Anordnung der Qubits zurückzuführen, die in IBM Q Experience visualisiert sind. $\bar{T}1$ bezeichnet die durchschnittliche Relaxationszeit der Qubits und ist die Zeit, in der ein Qubit von einem Zustand höherer Energie in den Zustand niedriger Energie wechselt und somit seinen Wert verändert. $\bar{T}2$ steht für die durchschnittliche Dekohärenzzeit der Qubits und gibt die Zeit an, nach der eine Superposition durch Umwelteinflüsse zerfällt [WZY05]. In beiden Fällen sind höhere Werte von Vorteil, da die gewünschten Eigenschaften länger bestehen bleiben. Die Werte stammen von IBM Q Experience [IBM] und Rigetti [Rigb]. Die besten Werte haben die Quantencomputer 2 und 3 von IBM, die im Vergleich zu den anderen beiden IBM-Modellen etwa ein halbes Jahr neuer sind. Der Quantencomputer von Rigetti weist auffällig geringere Werte auf, als die von IBM. Die Auslesegenauigkeit gibt an, in wie viel Prozent der Fälle das Auslesen von Qubits das richtige Ergebnis liefert. Die Werte der durchschnittlichen Auslesegenauigkeit von IBM ergeben sich aus den Fehlerraten der einzelnen

Qubits. Es fällt auf, dass die 5-Qubit-Quantencomputer eine bessere Auslesegenauigkeit aufweisen, als die Quantencomputer mit 14 und 16 Qubits, die eine Genauigkeit von etwa 93% haben. D-Wave macht keine Angaben zur Qubit-Qualität.

Merkmal	**Qiskit**	**Cirq**	**Forest**	**Strawberry Fields**	**QDK**	**Ocean**	**ProjectQ**	**PennyLane**	**XACC**
Entwickler	IBM	Google	Rigetti	Xanadu	Microsoft	D-Wave	ETH Zürich	Xanadu	Eclipse
Provider-agnostisch	×	×	×	×	×	×	✓	✓	✓
Programmier-sprache	Python	Python	Python	Blackbird (Python)	Q# (Python)	Python	Python	Python	Python
QC-Paradigma	gatterba-siert	gatterba-siert	gatterba-siert	gatterba-siert	topologisch	adiabatisch	gatterba-siert	gatterba-siert	gatterba-siert
Informations-einheit	Qubit	Qubit	Qubit	Qumode	Qubit	Qubit	Qubit	Qubit	Qubit
Simulatoren	✓	✓	✓	✓	✓	✓	✓	✓	✓
Quantencom-puter	1,2,3,4	×	5	×	×	6	1,2,3,4	1,2,3,4,5	1,2,3,4,5,6
Lizenz	Apache-2.0	Apache-2.0	Apache-2.0	Apache-2.0	MIT	Apache-2.0	Apache-2.0	Apache-2.0	EPL

Tabelle 4.1: Vergleichstabelle der untersuchten Frameworks.

ID	Bezeichnung	Hersteller	Paradigma	Technologie	Anzahl Qubits	Qubit Verknüpfung	$\bar{T}1$ (μs)	$\bar{T}2$ (μs)	Auslesegenauigkeit
1	ibmq_5_yorktown - ibmqx2	IBM	gatterbasiert	Supraleiter	5	2,4	54,76	44,81	98,52%
2	ibmq_vigo	IBM	gatterbasiert	Supraleiter	5	1,6	90,01	67,06	97,32%
3	ibmq_ourense	IBM	gatterbasiert	Supraleiter	5	1,6	87,86	62,99	96,42%
4	ibmq_16_melbourne	IBM	gatterbasiert	Supraleiter	14	2,71	47,63	64,99	93,46%
5	Aspen-4-16Q-A	Rigetti	gatterbasiert	Supraleiter	16	2,25	25,24	19,89	93,02%
6	D-Wave 2000Q	D-Wave	adiabatisch	Supraleiter	2048	-	-	-	-

Tabelle 4.2: Vergleichstabelle der Quantencomputer.

4.1 Qiskit

Qiskit wird von IBM und Abraham et al. [AAA+19] entwickelt und ist ein Python-Framework, das aus den fünf Modulen Terra, Aer, Aqua, Ignis und IBMQ-Provider besteht. Terra beinhaltet die Klassen, um einen Quantenalgorithmus zu programmieren und zu optimieren. Dazu zählen die Implementierungen der Qubits und Gatter, sowie die der Übersetzer. Das Aer-Modul beinhaltet verschiedene Simulatoren, um die programmierten Algorithmen lokal ausführen und testen zu können. Aqua ist eine Bibliothek, die verschiedene Algorithmen und Orakel beinhaltet, sowie Experimente in den Bereichen Chemie, künstliche Intelligenz (KI), Optimierung und Finanzen ermöglicht. Im Ignis-Modul befinden sich Experimente, um das Rauschen der Quantencomputer zu verstehen und zu beheben. Das fünfte Modul, IBMQ-Provider, ist optional und wird dazu benötigt, um die Quantenalgorithmen auf den Quantencomputern von IBM Q Experience auszuführen. Es beinhaltet unter anderem Funktionen, die den individuellen API-Schlüssel speichern bzw. abrufen und ein bestimmtes Backend auswählen. Fremde Backends werden in Qiskit derzeit nicht offiziell unterstützt. Allerdings werden auf der Website unter anderem Drittanbieter-Backends erwähnt und das Framework ist darauf ausgelegt zusätzliche Backends implementieren zu können [AAA+19].

4.2 Cirq

Cirq [Cir] ist das Quanten-Framework von Google. Es basiert auf Python und befindet sich aktuell in der Alpha, sodass in Zukunft mit deutlichen Erweiterungen und Änderungen zu rechnen ist. Die Funktionalität, die zum Schreiben von Algorithmen benötigt wird, ist bereits vorhanden. Es können Schaltkreise und Qubit-Register definiert werden. Die Schaltkreise werden aus den gewöhnlichen oder den Google-spezifischen Gattern aufgebaut und es ist möglich die Schaltkreise zu visualisieren. Das Ausführen der Algorithmen findet auf Simulatoren statt, wobei es für Schaltkreise, die aus den gewöhnlichen bzw. den Google-spezifischen Gattern bestehen, jeweils eigene Simulatoren gibt. Außerdem bietet Cirq unterstützende Funktionalität in den Bereichen Optimierung und Testen eines Algorithmus an, wie z. B. die Minimierung der Anzahl der Gatter. Cirq-Algorithmen können nur auf Simulatoren ausgeführt werden, da die Quantencomputer von Google nicht öffentlich verfügbar sind. Die benötigte Funktionalität ist in Cirq jedoch vorhanden. Um einen Cirq Algorithmus auf einem Quantencomputer auszuführen, gibt es die Möglichkeit ihn in OpenQASM [CBSG17] zu übersetzen. Der OpenQASM-Code kann z. B. auf der Hardware von IBM ausgeführt werden [Cir].

4.3 Forest SDK

Das Forest SDK [Riga] wird von Rigetti entwickelt. Forest besteht aus den drei Komponenten pyQuil, QuilC und QVM. pyQuil ist eine Bibliothek für Python, die es ermöglicht Quil-Programme in Python zu schreiben. Die Quantum Instruction Language (Quil) wird ebenfalls bei Rigetti von Smith et al. [SCZ17] entwickelt. Quil ist eine Sprache, um Instruktionen für Quantencomputer auf einem klassischen Computer zu formulieren. QuilC ist ein Quil-Compiler, der den pyQuil-Code in Maschinencode übersetzt, der auf Simulatoren und den Quantencomputern von Rigetti ausgeführt werden kann. Die Quantum Virtual Machine (QVM) ist ein Simulator, der als lokaler Server ausgeführt wird und die Quil-Programme verarbeiten kann [Riga].

4.4 Strawberry Fields

Strawberry Fields [KIQ+19] ist ein Framework für Continous-Variable Quantum Computing und wird von Xanadu entwickelt. Der größte Unterschied zum herkömmlichen Quantencomputing besteht darin, dass keine Qubits verwendet werden, sondern sogenannte Qumodes. Qumodes unterscheiden sich von Qubits unter anderem in den Zuständen, in denen sie sich befinden können, welche Gatter auf sie angewendet werden können und wie sie gemessen werden [KIQ+19]. Die Algorithmen werden in der Sprache Blackbird geschrieben. Blackbird ist an Python angelehnt und weist viele Gemeinsamkeiten auf. Ein großer Unterschied zu Python ist, dass Variablen statisch typisiert sind. Das fertige Blackbird-Programm wird anschließend kompiliert und auf dem definierten Backend ausgeführt. Als Backends stehen aktuell diverse Simulatoren zur Verfügung. In Zukunft sollen Photonen-basierte Quantencomputer von Xanadu als Backends verfügbar sein [KIQ+19].

4.5 Quantum Development Kit

Das QDK [Mic] wird von Microsoft entwickelt. Es besteht aus der Programmiersprache Q# inklusive Bibliotheken, diversen Backends, Erweiterungen für Visual Studio. Außerdem wird ein Python-Modul und die Erweiterung IQ# bereitgestellt, die das Ausführen von Q#-Operationen in Python ermöglichen. Die Quantenalgorithmen werden in Q# geschrieben. Um einen Algorithmus auszuführen, wird ein klassisches Programm in C# oder Python benötigt, welches als Treiber bezeichnet wird. Im Treiber können die Quantenalgorithmen importiert werden. Bevor ein Quantenalgorithmus ausgeführt werden kann, wird im Treiber ein Backend ausgewählt. Zur Auswahl stehen Simulator oder Estimator. Die anschließende Ausführung des Quantenalgorithmus findet asynchron statt. Die Quantenalgorithmen können nicht auf echten Quantencomputern ausgeführt werden, da Microsoft topologische Qubits verwendet und keine funktionierende Umsetzung eines topologischen Quantencomputers existiert [LaR19].

4.6 Ocean

Ocean [D-Wb] wird von D-Wave entwickelt und ist eine Sammlung diverser Tools, die das Programm für den Quantencomputer vorbereiten. D-Wave verwendet adiabatische Quantencomputer, wodurch keine Gatter bei der Programmierung zum Einsatz kommen. Stattdessen wird das Problem als mathematische Funktion formuliert. Mapping-Methoden übersetzen die Problembeschreibung in ein Binary Quadratic Model (BQM), die der Quantencomputer verarbeiten kann. Ein BQM ist eine mathematische Funktion, die für N gegebene binäre variablen von einem Quantencomputer minimiert wird. Um die Übersetzung zu erleichtern, bietet Ocean zwei Tools an, die die Übersetzung für Graphen- und Constraint-Satisfaction-Probleme übernehmen. Für andere Probleme kann Ocean um zusätzliche Mapping-Methoden erweitert werden. Eine weitere Aufgabe der Mapping-Methoden ist es die Ergebnisse des Quantencomputers zurückzuübersetzen. Im nächsten Schritt wird das BQM über die Uniform Sampler API konfiguriert. Anschließend wird das BQM auf einem Sampler bzw. Backend ausgeführt. Mögliche Backends sind Simulatoren, sowie die Quantencomputer von D-Wave. Ocean erlaubt außerdem die Programmierung eigener Backends [D-Wb].

4.7 ProjectQ

ProjectQ [Pro] wird an der ETH Zürich von Steiger et al. [SHT18] entwickelt. Das Framework beinhaltet eine in Python eingebettete DSL, einen modularen Compiler mit verschiedenen Kompilierschritten und diverse Backends. Der Algorithmus wird in der DSL geschrieben und an die Main-Engine übergeben. Die Main-Engine ist der Startpunkt des Compilers und in ihr wird vom Programmierer festgelegt, welche weiteren Compiler-Engines, z. B. zur Optimierung und Übersetzung, aufgerufen werden. Die letzte Compiler-Engine ist das Backend-Interface, welche den Code für das ausgewählte Backend vorbereitet. Als Backends bietet ProjectQ einen Simulator bzw. Emulator zum Testen, einen Ressource Estimator, um die benötigten Ressourcen abschätzen zu können und einen Circuit Drawer zum Darstellen des Schaltkreises. Um die Algorithmen auf echten Quantencomputern ausführen zu können, bietet ProjectQ aktuell das Backend von IBM Q Experience an. Der modulare Aufbau des Frameworks ermöglicht es neue Backends zu unterstützen. Steiger et al. planen ProjectQ regelmäßig um neue Bibliotheken, Compiler-Engines und Hardware-Backends zu erweitern [SHT18].

4.8 PennyLane

PennyLane [BIS+19] wird ebenfalls von Xanadu entwickelt und hat einen starken Fokus auf hybriden Berechnungen, die aus Quanten- und klassischen Komponenten bestehen. Dadurch ist es möglich die Komponenten beliebig miteinander zu verknüpfen, solange durch die Verknüpfung ein gerichteter azyklischer Graph entsteht, was in Abbildung 4.2 zu sehen ist [BIS+19]. Die Programme können auf den Simulatoren von PennyLane ausgeführt werden. Zusätzlich werden Plugins für Strawberry Fields, Forest, Qiskit, ProjectQ und Q# angeboten. Die Plugins mappen die Gatter der Frameworks auf die Gatter von PennyLane bzw. ergänzen fehlende Gatter. Dadurch kann der Nutzer die gewohnte PennyLane-Syntax verwenden und das Plugin übersetzt in das ausgewählte Framework. Aus diesem Grund können die Algorithmen auf den Backends der unterstützten Frameworks ausgeführt werden [BIS+19].

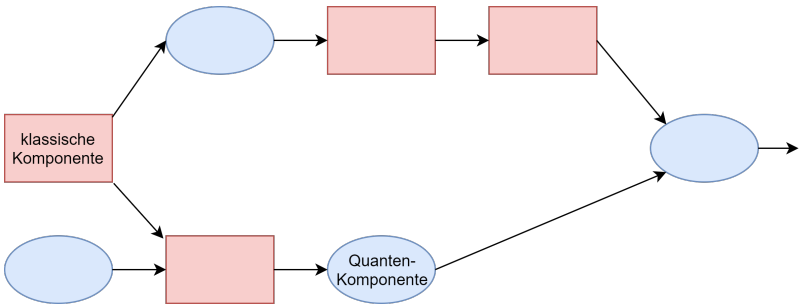


Abbildung 4.2: Kombination von Quanten- und klassischen Komponenten in einem gerichteten azyklischen Graph [BIS+19].

4.9 XACC

XACC [XAC] wird als Eclipse-Projekt entwickelt und ist ein erweiterbares Framework für die framework- und hardwareunabhängige Programmierung von Quantenalgorithmen [MDL+18]. Bereits vorhandene Plugins erlauben die Kompilierung von Kernel aus den Sprachen Quil und OpenQASM. Ein Kernel ist laut der Dokumentation die programmatische Repräsentation der unitären Instruktionen, die auf dem Qubit-Register durchgeführt werden. Wahlweise können die Kernel auch direkt in XACC geschrieben werden. Die Kompilierung wandelt die Kernel in eine Zwischenrepräsentation um, die selbst auch ein Kernel ist, aber durch einen weiteren Kompilierschritt auf einem der unterstützten Backends ausgeführt werden kann [MDLH18]. Die unterstützten Backends sind unter anderem die Quantencomputer bzw. Simulatoren von Rigetti, IBM und D-Wave [XAC].

5 Konzeption

In diesem Kapitel wird das Konzept einer Plattform vorgestellt, die Quantenalgorithmen in Form von Quanten-Services bereitstellen kann.

5.1 Problemstellung

Es gibt von jedem Hersteller von Quantencomputern ein Framework, um einen Quantenalgorithmus zu programmieren. Außerdem verfolgen die Hersteller teilweise unterschiedliche Paradigmen, um einen Quantencomputer aufzubauen. Es ist das Ziel dieser Arbeit eine Plattform zu entwickeln, die, unabhängig von dem verwendeten Framework und dem verfolgten Paradigma, Quanten-Services aus Quantenalgorithmen erstellen kann. Durch die Quanten-Services sollen die Quantenalgorithmen in der Cloud als QAaaS verfügbar gemacht werden. Außerdem sollen sie über eine programmiersprachenunabhängige API verwendbar sein. Dadurch können Entwickler klassischer Anwendungen Quantenalgorithmen verwenden, ohne die Implementierung kennen zu müssen. Außerdem wird die Wiederverwendbarkeit der Quantenalgorithmen erhöht.

5.2 Erkenntnisse

Durch die vorangegangene Analyse vorhandener Frameworks und Quantencomputer Paradigmen, können einige Erkenntnisse gesammelt werden.

Aus Kapitel 4 geht hervor, dass alle untersuchten Frameworks für die Programmiersprache Python zur Verfügung stehen bzw. eine Ausführung über Python ermöglichen, wie es beim QDK der Fall ist. Somit ist die Implementierung eines Quantenalgorithmus im Kontext dieser Ausarbeitung ein Python-Skript. Das bedeutet, dass der Service ebenfalls in Python programmiert werden kann, um eine Ausführung des Quantenalgorithmus nativ zu unterstützen.

Einen Algorithmus in der Art zu abstrahieren, sodass er für jedes Framework generiert werden kann, ist sehr aufwendig. Die Unterschiede der einzelnen Frameworks und der Quantencomputer-Paradigmen erfordern eine eigene Sprache bzw. ein eigenes Framework, um zwischen den vorhandenen Frameworks zu übersetzen. Des Weiteren muss darauf geachtet werden, dass z. B. bei den Frameworks für gatterbasierte Quantencomputer alle Gatter gleichermaßen unterstützt werden oder ein adäquates Mapping gefunden wird. Ein weiteres Problem ist das Mapping zwischen den Paradigmen. Siu [Siu05] zeigt, dass zumindest ein Mapping vom gatterbasierten Paradigma in das adiabatische Paradigma existiert. Ein eigenes Framework zu entwickeln, sowie Übersetzungen und Mappings für alle Frameworks und Paradigmen zu finden, übersteigt den Rahmen dieser Ausarbeitung.

Alternativ können die verschiedenen Framework-Implementierungen eines Algorithmus durch einen Container gruppiert werden, sodass ein Algorithmus mit mehreren Implementierungen umsetzbar ist, ohne ein Mapping vorzunehmen. Dieser Ansatz wird in dieser Arbeit verfolgt.

5.3 Anforderungen

Aus der Problemstellung und den Erkenntnissen können Anforderungen abgeleitet werden, die von der Plattform und den Quanten-Services erfüllt werden sollen.

5.3.1 Plattform

Eine Übersicht der Anforderungen ist in Tabelle 5.1 zu sehen. Die Plattform soll Quantenalgorithmen verwalten können, weshalb es eine Möglichkeit geben muss, Quantenalgorithmen zu speichern. Hierfür wird ein Repository benötigt. Um die Daten im Repository ablegen zu können, müssen diese an den Server gesendet werden können. Dies soll über zwei Wege möglich sein. Der erste Weg ist das Hochladen eines Python-Skripts, das den Code des Quantenalgorithmus enthält. Die zweite Möglichkeit ist die Benutzung des Editors auf der Plattform, der eine Programmierung des Algorithmus im Browser zulässt. Nach der Erstellung eines Quantenalgorithmus auf der Plattform soll er ebenso bearbeitet und gelöscht werden können. Außerdem sollen Quantenalgorithmen, die den gleichen Algorithmus implementieren und sich nur in der Wahl des Frameworks unterscheiden, gruppiert werden können. Das hat den Vorteil, dass dadurch ein gebündelter Export möglich ist, was für die letzte Anforderung, den Export von Quanten-Services, wichtig ist. Außerdem soll es möglich sein die Algorithmen zu parametrisieren, sodass Eingabeparameter definiert werden können und der Algorithmus für verschiedene Eingaben funktioniert. Die Plattform soll die hochgeladenen Algorithmen in Form von Quanten-Services exportieren können, die mit geringem Aufwand in der Cloud verfügbar gemacht werden können, um eine plattform- und programmiersprachenunabhängige Nutzung und Wiederverwendbarkeit zu gewährleisten. Daraus ergeben sich spezielle Anforderungen an die Quanten-Services, welche im nachfolgenden Abschnitt besprochen werden.

Anforderung	**Beschreibung**
Repository für Quantenalgorithmen	Die Plattform muss ein Repository beinhalten, um Quantenalgorithmen speichern zu können.
Verwaltung von Quantenalgorithmen	Die Plattform muss Funktionen anbieten, um Quantenalgorithmen erstellen bzw. hochladen, bearbeiten und löschen zu können.
Gruppierung von Implementierungen	Die verschiedenen Implementierungen des gleichen Quantenalgorithmus sollen gruppiert werden können.
Parametrisierung von Quantenalgorithmen	Quantenalgorithmen sollen parametrisierbar sein, damit sie mit verschiedenen Eingabewerten funktionieren.
Export von Quanten-Services	Die Plattform muss Quantenalgorithmen als Quanten-Services exportieren können.

Tabelle 5.1: Übersicht der Anforderungen an die Plattform.

5.3.2 Quanten-Service

In Tabelle 5.2 sind die Anforderungen an die Quanten-Services zusammengefasst. Um eine programmiersprachenunabhängige Verwendung der Quantenalgorithmen zu ermöglichen, soll der Quanten-Service eine REST-Schnittstelle anbieten, die die Funktionalität nach außen verfügbar macht und gleichzeitig die Implementierungsdetails verschleiert. Das ermöglicht eine Verwendung der Quantenalgorithmen innerhalb aller verbreiteten Programmiersprachen, da es in der Regel immer möglich ist HTTP-Anfragen zu versenden und HTTP-Antworten zu verarbeiten. Außerdem ist es von Vorteil die Quanten-Services in Python zu entwickeln, weil sämtliche Frameworks in Python verwendbar sind. Das ermöglicht eine native Ausführung der Quantenalgorithmen innerhalb der Schnittstelle. Des Weiteren sollte ein Quanten-Service alle Informationen zu den benötigten Abhängigkeiten beinhalten. Die Abhängigkeiten sind aufgrund der verwendeten Frameworks und Module innerhalb der Quantenalgorithmen variabel und müssen spezifiziert sein, um sie bei der Bereitstellung in der Cloud automatisiert installieren zu können.

Anforderung	**Beschreibung**
REST-Schnittstelle	Der Quanten-Service soll eine REST-Schnittstelle anbieten, um die plattform- und programmiersprachenunabhängige Nutzung zu ermöglichen.
Python als Programmiersprache	Für die Implementierung der Quanten-Services soll die Programmiersprache Python verwendet werden.
Abhängigkeiten	Alle Informationen zu den benötigten Abhängigkeiten sollen im Quanten-Service vorhanden sein.

Tabelle 5.2: Übersicht der Anforderungen an die Quanten-Services.

5.4 Architektur

Aus den zuvor definierten Anforderungen lässt sich eine Architektur für die Plattform und die Quanten-Services erstellen. Wie in Abbildung 5.1 zu sehen ist, umfasst die Plattform einen Webservice, der für die Verarbeitung von HTTP-Anfragen und die Bereitstellung der HTML-Seiten zuständig ist. Über den Webservice können die Nutzer die Quantenalgorithmen hochladen und verwalten. Des Weiteren umfasst die Plattform ein Repository, in dem die Quantenalgorithmen gespeichert und abgerufen werden können. Die Plattform dient dazu, Quantenalgorithmen in Form eines Quanten-Service zu generieren. Der Quanten-Service stellt eine REST-Schnittstelle bereit, die HTTP-Anfragen verarbeitet und den Quantenalgorithmus ausführt. Dabei werden die Implementierungsdetails verborgen und eine Programmiersprachenunabhängigkeit gewährleistet. Führt der Quanten-Service den enthaltenen Quantenalgorithmus aus, werden die programmierten Instruktionen über eine im verwendeten Framework definierte Schnittstelle an das jeweilige Quanten-Backend gesendet. Ein Quanten-Backend ist beispielsweise ein Quantencomputer oder Simulator. Nach der Berechnung wird das Ergebnis zurückgegeben.

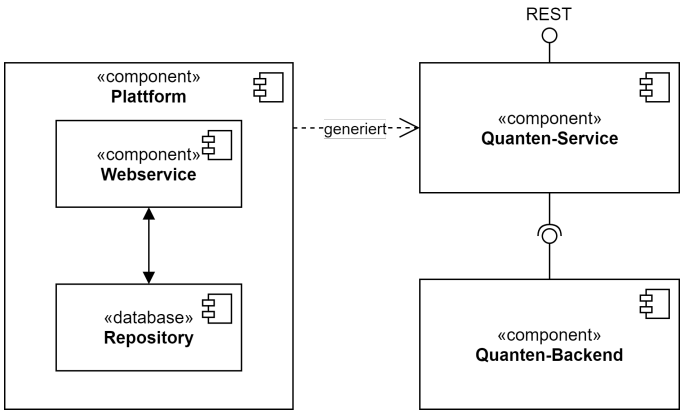


Abbildung 5.1: Architektur einer Plattform, die Quanten-Services bereitstellt.

5.5 Empfohlene Praktiken

In diesem Abschnitt werden empfohlene Praktiken für Quanten-Services vorgestellt, die aus den Anforderungen und der Architektur abgeleitet werden. Diese Praktiken sollten generell bei der Umsetzung von Quanten-Services beachtet werden.

5.5.1 Abhängigkeitsdatei

Ein Quanten-Service unterscheidet sich von anderen Quanten-Services in erster Linie durch den Algorithmus, der für verschiedene Frameworks implementiert sein kann. Daraus resultieren variable Abhängigkeiten. Damit der Quanten-Service in einer neuen Umgebung verwendet werden kann, müssen die benötigten Abhängigkeiten bekannt sein. Diese können bei der Generierung in einer Abhängigkeitsdatei aufgeführt und dem Quanten-Service beigelegt werden, sodass die neue Umgebung die benötigten Abhängigkeiten installieren kann.

5.5.2 Statisch Typisierte Eingabeparameter

Die dynamische Typisierung hat den Vorteil, dass einer Variable flexibel ein Wert eines beliebigen Datentyps zugeordnet werden kann. Dadurch treten Fehler, die durch die Verwendung eines falschen Typs entstehen können, erst zur Laufzeit der Anwendung auf und müssen im Code behandelt werden. Um sicherzustellen, dass es während der Laufzeit eines Quantenalgorithmus zu keinen Fehler aufgrund des Datentyps der Eingabeparameter kommt, können diese statisch typisiert werden. Bei der Erstellung eines Algorithmus auf der Plattform ist darauf zu achten, dass die Parameter mit einem bestimmten Datentyp definiert werden. Wird aus dem Algorithmus ein Quanten-Service generiert, hat die Schnittstelle die Aufgabe den Quantenalgorithmus mit den korrekten Datentypen für die Eingabeparameter aufzurufen. Werden die falschen Datentypen in der Anfrage übergeben, kann eine Konvertierung versucht werden. Ansonsten sollte die Anfrage mit einem entsprechenden Hinweis abgelehnt werden.

5.5.3 Einheitliche Startmethode

Bei der Programmierung eines Quantenalgorithmus kann die Logik in mehrere Methoden aufgeteilt werden, die unterschiedlich und frei benannt sind. Wird aus diesem Algorithmus ein Quanten-Service generiert, ist es nicht unbedingt möglich den Startpunkt der Berechnung festzustellen. Damit dies zuverlässig möglich ist, soll eine Startmethode festgelegt und in jedem Quantenalgorithmus einheitlich verwendet werden. In der Programmierung wird der Startmethode oft mit main benannt. Der Startmethode werden sämtliche Parameter übergeben. Innerhalb der Methode können weitere Methoden aufgerufen werden.

5.5.4 Universelle Schnittstelle

Die Plattform soll Quanten-Services generieren, die plattform- und programmiersprachenunabhängig von einer beliebigen klassischen Anwendung verwendet werden können. Damit das möglich ist, müssen die Quanten-Services eine universelle Schnittstelle anbieten, die von jeder Programmiersprache implementiert werden kann. Die verbreiteten Programmiersprachen unterstützten es HTTP-Anfragen zu versenden und zu empfangen. Daher kann die Schnittstelle der Quanten-Services so implementiert werden, dass sie dieses Protokoll für die Kommunikation verwendet.

5.5.5 Universelles Antwortformat

Die Ergebnisse, die von Quantencomputern und Simulatoren zurückgeliefert werden, können sich in den verwendeten Datentypen und -strukturen unterscheiden. Außerdem kann innerhalb des Quantenalgorithmus eine Verarbeitung der Ergebnisse stattfinden. Es muss sichergestellt werden, dass die Schnittstelle die Daten in einem standardisierten Format versendet, sodass sie plattform- und programmiersprachenunabhängig in den klassischen Anwendungen verarbeitet werden können. Verbreitete Datenformate, die in durch die gängigen Programmiersprachen unterstützt werden und über HTTP versendet werden können, sind unter anderem JSON und XML. Die Ausgabe des Quantenalgorithmus muss für das jeweilige Format serialisierbar sein.

6 Implementierung

In diesem Kapitel wird eine prototypische Implementierung des Konzepts aus Kapitel 5 vorgestellt. Hierfür wird die Quantum Service Platform (QUASP) präsentiert, die zur Verwaltung von Quantenalgorithmen und zum Export von Quanten-Services genutzt werden kann. In den folgenden Abschnitten werden die Komponenten von QUASP vorgestellt und die Benutzung anhand eines Anwendungsbeispiels demonstriert.

6.1 Quantum Service Platform

Die Plattform besteht aus einem Webservice und einem Repository, wie in der Architektur aus Abbildung 5.1 zu sehen ist. In diesem Abschnitt wird beschrieben, wie die zwei Komponenten umgesetzt sind.

Der Webservice ist als Django-Anwendung implementiert. Django [Dja] ist ein Python-Framework, dass Entwickler bei der Umsetzung von Webanwendungen unterstützt. Es beinhaltet Funktionalitäten, um URLs zu definieren und mit Views zu verknüpfen. Views sind Python-Funktionen und führen die Logik hinter einer URL aus, wie z. B. das Abfragen von Datenbankelementen. Außerdem bietet Django eine Templating-Engine, wodurch die HTML-Seiten dynamisch befüllt werden können. Der Webservice dient als Benutzeroberfläche und Schnittstelle zum Repository, sodass die Algorithmen verwaltet und die Quanten-Services exportiert werden können. Die konkreten Funktionen werden in Abschnitt 6.3 anhand eines Anwendungsbeispiels erklärt.

Das Repository wird durch eine SQLite-Datenbank realisiert. Die Datenbank wird bei der Erstellung eines Django-Projekts erzeugt und mithilfe von Django verwaltet. In einem Python-Skript wird das Datenbankschema festgelegt, welches in Abbildung 6.1 als Entity-Relationship-Diagramm visualisiert ist. Es gibt die drei Entitäten Algorithmus, Implementierung und Parameter, welche durch die Quadrate dargestellt sind. Die Entität Algorithmus fungiert als Container für beliebig viele Implementierungen. Sie hat lediglich zwei Attribute für Name und Beschreibung, die durch die Ellipsen repräsentiert werden. Gestrichelte Attribute, wie die ID und das Erstellungsdatum, werden automatisch generiert. Mit einem Algorithmus sind beliebig viele Implementierungen und Parameter assoziiert, sodass beide Entitäten einen Fremdschlüssel AlgorithmusId besitzen, der einen bestimmten Algorithmus identifiziert. Ein Parameter spezifiziert die Eingabe für einen Algorithmus und beinhaltet Name, Datentyp und eine Beschreibung. Dadurch wird festgelegt, welche Daten an die REST-Schnittstelle der Quanten-Services geschickt werden müssen. Eine Implementierung ist die konkrete Umsetzung des Algorithmus. Sie besitzen ebenfalls Eingabefelder für einen Namen und eine Beschreibung. Außerdem werden für Implementierungen das Framework und der Quellcode angegeben. Das Framework ist wichtig, um die Abhängigkeiten für den Quanten-Service spezifizieren zu können. Der Quellcode wird später durch den Quanten-Service ausgeführt und muss die Parameter, die mit demselben Algorithmus assoziiert sind, wie die Implementierung, als Eingabe erwarten.

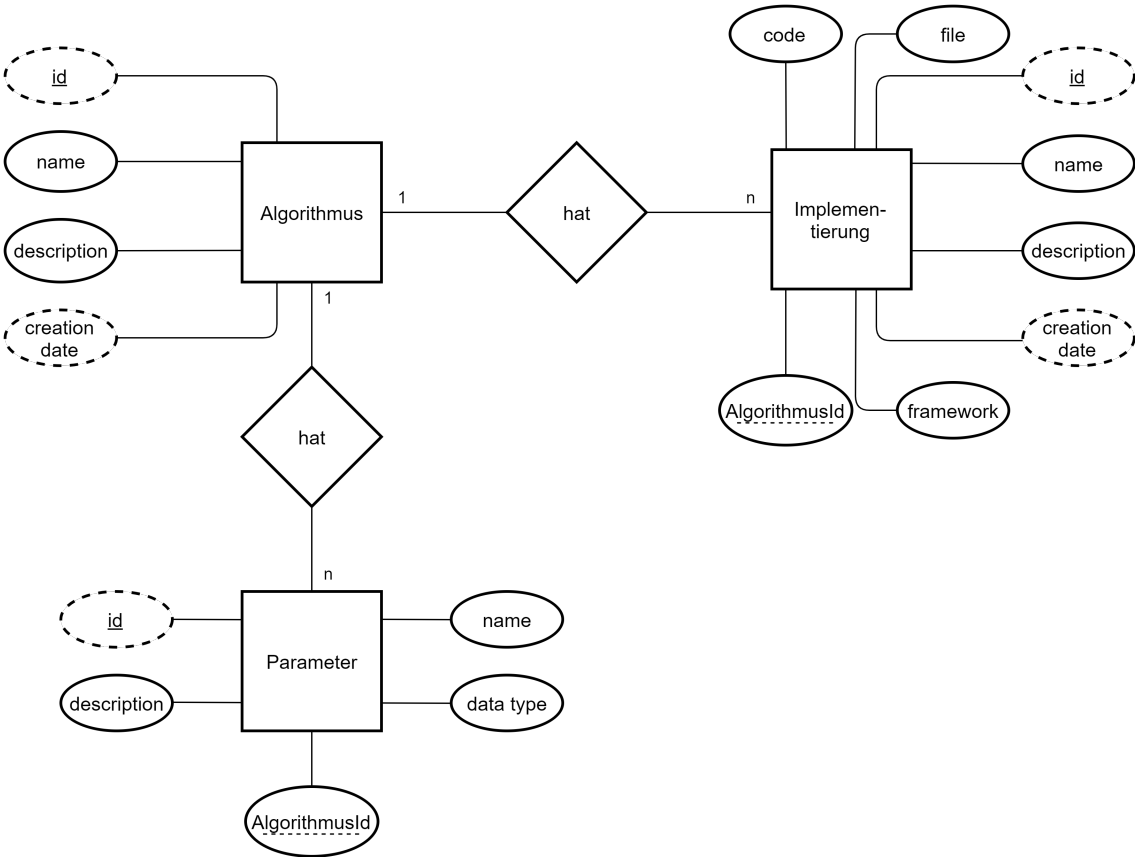


Abbildung 6.1: Entity-Relationship-Diagramm des Datenbankschemas.

6.2 Quanten-Services

Ein Quanten-Service ist ein klassischer Service, der einen Quantenalgorithmus beinhaltet. Über eine REST-Schnittstelle kann der Quantenalgorithmus mit den übermittelten Eingabewerten ausgeführt werden.

Quanten-Services werden als Flask-Apps realisiert. Flask [Fla] ist, ähnlich wie Django, ein Python-Framework für die Entwicklung von Webservices, ist jedoch leichtgewichtiger. In Abbildung 6.2 sind die Komponenten dargestellt, die zu einem Quanten-Service gehören. Das Python-Skript app.py ist der Startpunkt der Flask-App und beinhaltet den Code der REST-Schnittstelle. Der Quantenalgorithmus, beispielhaft als algorithmus.py bezeichnet, ist ebenfalls in Python implementiert, wodurch er nativ durch die Schnittstelle ausgeführt werden kann. In der requirements.txt sind die Abhängigkeiten definiert, die in einer neuen Python-Umgebung installiert sein müssen, damit der Quanten-Service ausgeführt werden kann. Im Templates-Ordner befinden sich die HTML-Seiten, um den Algorithmus im Browser nutzen zu können und dienen zu Testzwecken, sowie als Beispiel einer Anwendung, die die Schnittstelle implementiert.

In QUASP werden die Quanten-Services aus einem Template generiert. Wird die Export-Funktion aufgerufen, werden die erforderlichen Daten aus der Datenbank abgerufen, aggregiert und in das Template kopiert. Anschließend wird das Python-Skript des Quantenalgorithmus in den Ordner des Quanten-Service kopiert. Der Quanten-Service wird als Zip-Datei zum Download bereitgestellt.

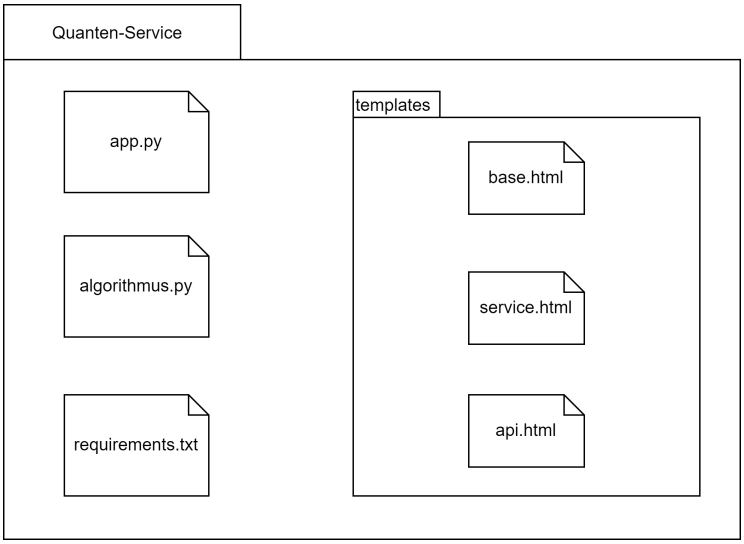


Abbildung 6.2: Paketdiagramm eines Quanten-Service.

6.3 Anwendungsbeispiel

In diesem Abschnitt wird anhand eines Anwendungsbeispiels die Funktionsweise von QUASP erklärt. Das Beispiel ist in Abbildung 6.3 abgebildet und zeigt einen Akteur, der QUASP verwendet, um einen Algorithmus mit einer Implementierung zu erstellen. Anschließend wird der Algorithmus als Quanten-Service exportiert. Der exportierte Quanten-Service wird beispielsweise in der Cloud gestartet. Klassische Anwendungen können nun Anfragen an die REST-Schnittstelle des Quanten-Service senden. Dieser startet den enthaltenen Quantenalgorithmus, welcher wiederum Instruktionen an ein Backend, das z. B. ein Simulator oder Quantencomputer sein kann, sendet. Die Instruktionen werden vom Backend ausgeführt und es findet eine Berechnung statt. Das Ergebnis der Berechnung wird an den Quanten-Service zurückgegeben, welcher mit dem Ergebnis auf die ursprüngliche Anfrage der klassischen Anwendung antwortet. Die klassische Anwendung kann das Ergebnis eines Quantenalgorithmus verwenden, ohne etwas über die Implementierung wissen zu müssen.

Das Beispiel aus Abbildung 6.3 wird im Folgenden am Beispiel des Shor-Algorithmus durchgeführt und illustriert. Zunächst muss in QUASP ein Algorithmus angelegt werden. Dies geschieht über das Formular aus Abbildung 6.4. Es wird ein Algorithmus mit dem Namen Shor´s Algorithm angelegt, welcher eine Beschreibung und den Parameter number spezifiziert. Ein neuer Parameter kann über den -Button angelegt werden, wodurch das Formular aus Abbildung 6.5 geöffnet wird. Dieses Formular erwartet einen Namen, eine Beschreibung und die Angabe eines Datentypen für den Parameter. Als Datentypen können Number, String und Boolean ausgewählt werden. Number repräsentiert hierbei ganze und Gleitkommazahlen. Bei der Wahl eines Namen ist darauf zu achten,

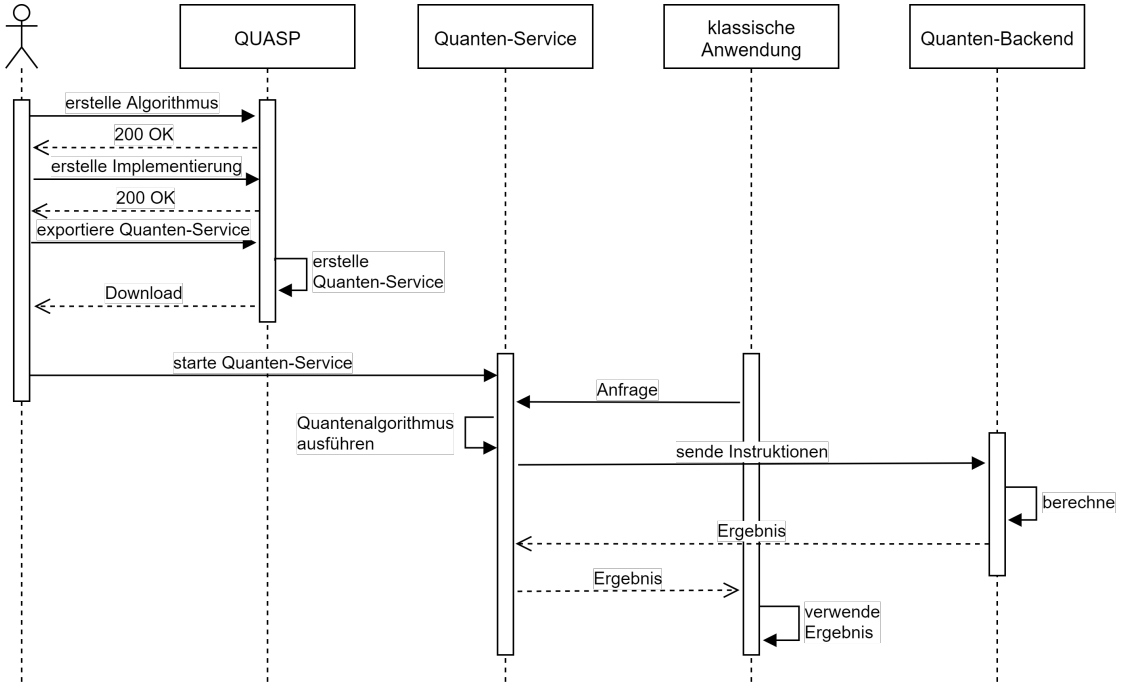


Abbildung 6.3: Das Sequenzdiagramm eines Anwendungsbeispiels.

dass er mit der Python-Syntax konform ist, da er als Bezeichner im Code verwendet wird. Deshalb ist eine Validierung eingebaut, die nur Namen zulässt, welche mit einem Buchstaben beginnen und nur aus Buchstaben, Zahlen und Unterstrichen bestehen.

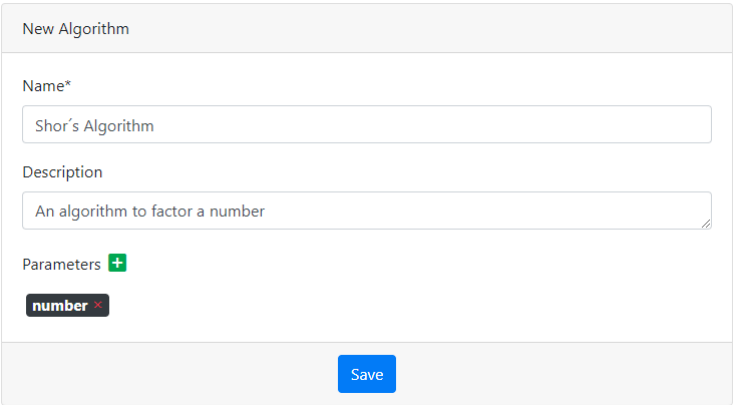


Abbildung 6.4: Das Formular zum Anlegen eines neuen Algorithmus in QUASP.

Nachdem der Algorithmus angelegt ist, muss eine Implementierung für diesen Algorithmus erstellt werden. Hierfür wird eine Qiskit-Implementierung des Shor-Algorithmus verwendet. Die Implementierung ist im Codebeispiel 6.1 zu sehen und besteht aus einer main-Methode, die den zuvor spezifizierten Parameter als Eingabe erwartet. In der Methode wird ein Backend, in diesem Fall ein Simulator, bestimmt. Anschließend wird der Shor-Algorithmus mithilfe von Qiskit Aqua erstellt. Aqua erstellt den Schaltkreis basierend auf den übergebenen Parametern. Der entstandene Schaltkreis wird auf dem Simulator ausgeführt und das Ergebnis ausgegeben. Der Python-Code

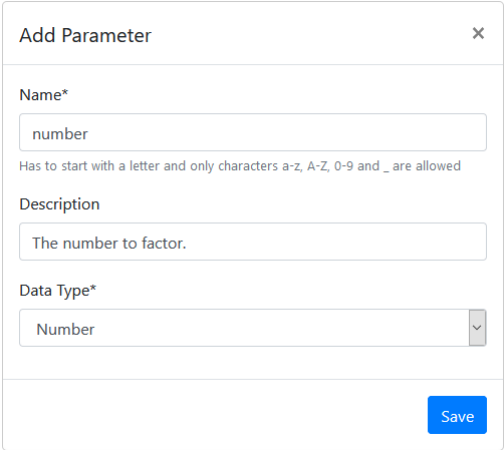


Abbildung 6.5: Das Formular zum Definieren eines Parameters in QUASP. Hier wird ein Parameter number für den Shor-Algorithmus erstellt.

Listing 6.1 Qiskit-Implementierung des Shor-Algorithmus

```
from qiskit.aqua.algorithms.single_sample.shor.shor import Shor
from qiskit import Aer


def main(number):
    simulator = Aer.get_backend('qasm_simulator')

    algorithm = Shor(N=number, a=2)
    result = algorithm.run(simulator)

    factors = result["factors"]
    return "The factors for " + str(number) + " are: " + str(factors)
```

kann bei der Erstellung einer neuen Implementierung, wie es in Abbildung 6.6 zu sehen ist, als Datei hochgeladen oder in den Editor kopiert werden. Zusätzlich muss angegeben werden, welchem Algorithmus diese Implementierung zugeordnet wird. In diesem Fall ist das Shor´s Algorithm. Die Implementierung wird Shor_Q gennant, da das verwendete Framework Qiskit ist. Als Framework können alle in dieser Arbeit vorgestellten Frameworks ausgewählt werden, sowie sämtliche Plugins von PennyLane. Optional kann eine Beschreibung angegeben werden. Es ist darauf zu achten, dass im Code eine main-Methode existiert, die alle Parameter als Eingabe erwartet, die durch den Algorithmus spezifiziert werden. Ansonsten gibt die Formular-Validierung eine Fehlermeldung aus. Dadurch wird sichergestellt, dass jede erstellte Implementierung einen einheitlichen Startpunkt für die Berechnung definiert und alle spezifizierten Parameter erwartet, sodass es beim späteren Export zu keinen Konflikten kommt.

Nach der Erstellung eines Algorithmus bzw. einer Implementierung, können diese in den jeweiligen Übersichten in den Abbildungen 6.7 und 6.8 gefunden werden. In der Übersicht für Algorithmen ist es möglich vorhandene Algorithmen zu bearbeiten und zu löschen. Bei den Implementierungen ist

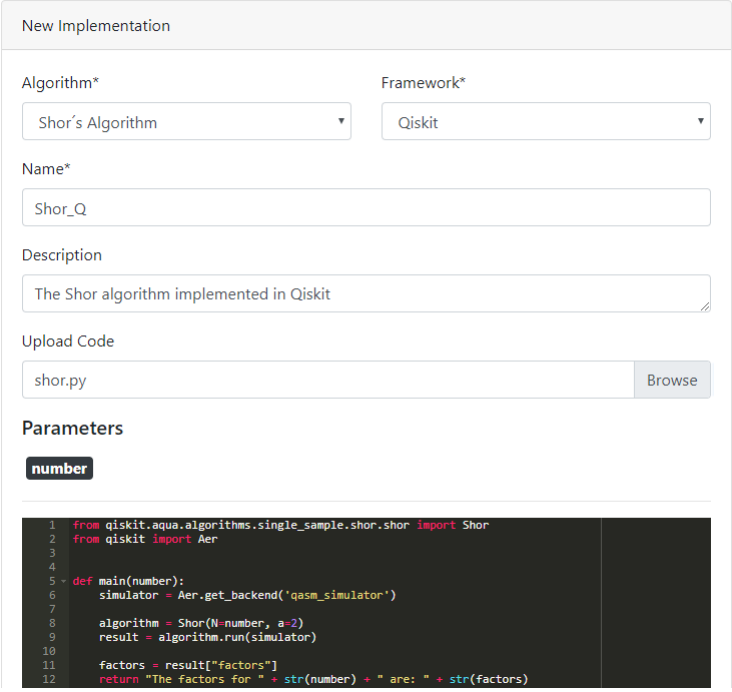


Abbildung 6.6: Formular für die Erstellung einer neuen Implementierung. Es müssen der zugehörige Algorithmus, das verwendete Framework und ein Name spezifiziert werden. Außerdem kann eine Beschreibung angegeben werden. Der Quellcode kann als Datei hochgeladen oder in den Editor eingegeben werden.

es zudem möglich den hochgeladenen Code zu betrachten, um ihn kopieren und wiederverwenden zu können. In beiden Übersichten kann über den -Button ein Export von Quanten-Services vorgenommen werden. Es gibt drei verschiedene Arten Quanten-Services zu exportieren. Über den -Button in der Algorithmen-Übersicht aus Abbildung 6.7 kann ein einzelner Quanten-Service exportiert werden, der alle Implementierungen des Algorithmus enthält. In der Implementierungs-Übersicht kommt der -Button zweimal vor. Zum einen gibt es den Button jeweils einzeln hinter den Implementierungen, wodurch ein Quanten-Service exportiert werden, der nur die jeweilige Implementierung beinhaltet. Zum anderen gibt es die Möglichkeit die komplette Liste der Implementierungen eines Algorithmus zu exportieren. In diesem Fall werden für alle Implementierungen des Algorithmus eigenständige Quanten-Services exportiert, die in einer Zip-Datei zum Download bereitgestellt werden.

Ein exportierter Quanten-Service, kann lokal oder in einer Cloud ausgeführt werden. Für dieses Beispiel wird der Quanten-Service des Shor-Algorithmus in Microsoft Azure [Azu] bereitgestellt. Die Bereitstellung in Azure funktioniert weitestgehend automatisch. Nach dem Upload wird automatisch nach einer requirements.txt gesucht und die angegebenen Abhängigkeiten installiert. Eine Flask-App wird ebenfalls automatisch erkannt, solange das Python-Skript standardmäßig benannt ist. Ansonsten muss das Start-Skript manuell definiert werden. Jedoch wird manuelles eingreifen benötigt, falls die Python-Abhängigkeiten zusätzliche Programme benötigen, um installiert werden zu können.
In der Prototyp-Implementierung sind die Quanten-Services mit einer Benutzeroberfläche ausgestattet. Sie simuliert eine klassische Anwendung, die die REST-Schnittstelle des Quanten-Service

Algorithms

Create

Name	Description	Creation Date	Options
Shor´s Algorithm	An algorithm to factor a number	12.10.2019 08:19	
Grover 3-SAT	The Grover Algorithm for solving the 3-SAT problem	07.09.2019 09:05	

Abbildung 6.7: Übersicht aller Algorithmen in QUASP. Es ist das Bearbeiten, Löschen und Exportieren von Algorithmen möglich.

Implementations

Create

Shor´s Algorithm Implementations

ID	Name	Description	Framework	Creation Date	Options
25	Shor_F	The Shor algorithm implemented in Forest	Forest SDK	15.10.2019 08:51	
24	Shor_C	The Shor algorithm implemented in Cirq	Cirq	15.10.2019 08:49	
23	Shor_Q	The Shor algorithm implemented in Qiskit	Qiskit	12.10.2019 08:19	

Grover 3-SAT Implementations

Abbildung 6.8: Übersicht aller Implementierungen in QUASP. Es gibt die Möglichkeit den Code einzelner Implementierungen zu betrachten, sie zu bearbeiten und zu löschen, sowie Quanten-Services zu exportieren.

implementiert. Wie in Abbildung 6.9 zu sehen ist, bietet die Benutzeroberfläche ein Formular zur Eingabe der spezifizierten Parameter an. Dabei werden die Eingabefelder abhängig vom spezifizierten Datentyp des Parameters generiert. In diesem Beispiel hat der Parameter number den Datentyp Number, weshalb ein Zahlenfeld als Input generiert wird. Für Strings wird ein Textfeld und für Wahrheitswerte eine Checkbox generiert. Sollten verschiedene Implementierungen in einem Quanten-Service vorhanden sein, können diese über ein Dropdown-Menü ausgewählt werden.
In diesem Beispiel soll die Zahl 15 mit dem Shor-Algorithmus faktorisiert werden. Wird die Eingabe mit dem Submit-Button bestätigt, wird eine Anfrage mit den Parametern an den Quanten-Service gesendet. Dieser führt den Quantenalgorithmus mit den jeweiligen Parametern aus, welcher wiederum Instruktionen für ein Backend generiert. Das Backend führt diese Instruktionen aus und gibt das Ergebnis bis zur klassischen Anwendung zurück. Das Ergebnis kann von der klassischen Anwendung für weitere Berechnungen verwendet oder wie in diesem Beispiel über die Benutzeroberfläche ausgegeben werden. Der Quantenalgorithmus hat korrekt die Faktoren 3 und 5 für die Zahl 15 identifiziert.

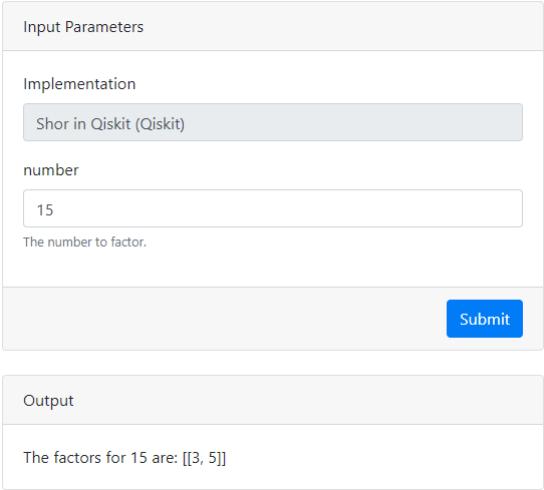


Abbildung 6.9: Die Benutzeroberfläche für den Quanten-Service besteht aus einem Formular, das Eingabefelder für die zu verwendende Implementierung und die spezifizierten Parameter des Algorithmus bereitstellt. Sobald ein Ergebnis des Quantenalgorithmus vorliegt, wird es unterhalb angezeigt.

Anhand dieses Anwendungsbeispiels wird deutlich, dass das erarbeitete Konzept umsetzbar ist und funktioniert. Die präsentierte Plattform erstellt Quanten-Services für alle vorgestellten Frameworks und Quantencomputer-Paradigmen. Die Services können in einer Cloud als QAaaS bereitgestellt werden und eine klassische Anwendung kann auf die Schnittstelle zugreifen, um eine Quantenberechnung auszuführen und ein Ergebnis zu erhalten.
Jedoch hat die Umsetzung einige Nachteile. Ein Algorithmus muss mehrmals implementiert werden, um ihn für verschiedene Provider zu verwenden. Eine universelle Implementierung ist nicht möglich, außer es wird ein Provider-agnostisches Framework verwendet, das bereits mehrere Provider unterstützt. Des Weiteren sind die Quanten-Services in Python programmiert, weshalb sie Programmiersprachen-abhängig bezüglich der Quanten-Frameworks sind. Quantenalgorithmen, die in einem Framework für eine andere Programmiersprache geschrieben sind, können über die Plattform nicht als funktionierende Quanten-Services exportiert werden. Das kann zu Problemen führen, falls zukünftige Frameworks nicht in Python implementiert sind. Für die in dieser Arbeit vorgestellten Frameworks ist diese Lösung jedoch zufriedenstellend.
Außerdem sind die Datentypen der Eingabeparameter auf Zahlen, Strings und Wahrheitswerte beschränkt. Jedoch können mit diesen Typen bereits viele Eingaben realisiert werden.
Darüber hinaus ist die Integration für Q# nicht vollständig. Es ist möglich ein Python-Skript auf die Plattform hochzuladen, das die Logik zur Ausführung der Q#-Operationen beinhaltet. Mehrere Dateien mit einer Implementierung zu verknüpfen, ist jedoch nicht möglich, sodass die benötigten Q#-Dateien noch nicht neben dem Python-Skript abgelegt werden können. Die Q#-Dateien können nachträglich in den exportierten Quanten-Service kopiert werden.

7 Zusammenfassung und Ausblick

Diese Arbeit stellt Quantencomputer Paradigmen vor und vergleicht existierende Frameworks zur Erstellung von Quantenalgorithmen. Die ausgewählten Frameworks erfüllen die Kriterien freie Verfügbarkeit und gute Dokumentation, damit jeder sie verwenden kann. Die Frameworks weisen alle eine ähnliche Architektur auf. Ein Quantenalgorithmus wird in einer klassischen Programmiersprache, wie z. B. Python, programmiert und mithilfe eines Compilers in Maschinencode übersetzt, der von einem Backend, wie einem Quantencomputer, verarbeitet werden kann. Das Ergebnis wird zurück an das Programm gesendet. Nahezu alle untersuchten Frameworks sind für Python verfügbar. Die einzige Ausnahme stellt Microsofts QDK dar, das Q# als Programmiersprache verwendet. Allerdings können die Q#-Operationen in Python importiert und ausgeführt werden. Für alle Frameworks existieren Simulatoren, jedoch bieten Cirq, Strawberry Fields und QDK aktuell keine echten Quantencomputer als Backend an. Ansonsten können die Provider-agnostischen Frameworks hervorgehoben werden, die durchschnittlich mehr Quantencomputer als Backends unterstützen und somit mehr Flexibilität bieten. Von den Provider-spezifischen Frameworks bietet Qiskit die meisten Quantencomputer als Backends an.

Die gewonnenen Erkenntnisse aus den vorangegangenen Untersuchungen dienen zur Formulierung von Anforderungen an die Plattform und Quanten-Services. Für die Plattform sind unter anderem ein Repository, sowie eine Möglichkeit zur Parametrisierung der Algorithmen notwendig. Die Quanten-Services sind in Python umgesetzt und bieten eine REST-Schnittstelle an. Aus den Anforderungen wird eine Architektur für die Plattform und Quanten-Services entworfen. Außerdem werden Praktiken für Quanten-Services empfohlen, wie z. B. die Angabe aller Abhängigkeiten in einer Datei, um eine Veröffentlichung in der Cloud automatisch durchführen zu können.

Schließlich wird eine prototypische Umsetzung des zuvor erarbeiteten Konzepts vorgestellt. Die Quantum Service Platform (QUASP) erfüllt die spezifizierten Anforderungen und kann Quantenalgorithmen als Quanten-Services bereitstellen. Sie ist eine Webanwendung, die mithilfe von Django realisiert ist und ein Repository beinhaltet, welches für das Speichern und Verwalten von Quantenalgorithmen verwendet wird. Die erstellten Quantenalgorithmen können von QUASP als Quanten-Services exportiert werden. Die Quanten-Services setzen ebenfalls die spezifizierten Anforderungen um. Sie sind als Flask-App implementiert und beinhalten eine Datei mit den benötigten Abhängigkeiten. Dadurch können sie leicht als Services in der Cloud bereitgestellt werden. Außerdem verfügen sie über eine REST-Schnittstelle, damit sie plattform- und programmiersprachenunabhängig von einer beliebigen Anwendung genutzt werden können. Die Funktionen von QUASP und der Quanten-Services werden anhand eines Anwendungsbeispiels demonstriert. Für das Beispiel wird ein Algorithmus inklusive Implementierung in QUASP erstellt und ein Quanten-Service exportiert. Anschließend wird der Quanten-Service über Microsoft Azure bereitgestellt und über eine klassische Anwendung verwendet.

Die präsentierte Umsetzung hat jedoch einige Nachteile. Ein Algorithmus muss für jedes Framework separat implementiert werden. Eine universelle Implementierung ist nicht möglich. Außerdem werden nur Frameworks unterstützt, die für Python angeboten werden. Andere Frameworks können mit der Plattform nicht verwendet werden, da die generierten Services auf Python basieren. Des Weiteren sind die Datentypen der Eingabeparameter auf Zahlen, Strings und Wahrheitswerte beschränkt. Außerdem ist Q# nicht vollständig integriert. Die Plattform unterstützt nur das Hochladen einer Datei, dem Python-Skript. Die Q#-Dateien können nachträglich in das Verzeichnis des Quanten-Service kopiert werden.

Ausblick

Durch QAaaS können Entwickler in der Zukunft von Quantenalgorithmen profitieren, ohne sich mit Quantencomputern und ihrer Programmierung auskennen zu müssen. Eine universelle Schnittstelle erlaubt es klassischen Anwendungen einen Quantenalgorithmus in der Cloud aufzurufen und das Ergebnis der Quantenberechnung zu verwenden. Dadurch können in Zukunft klassische Anwendung von einem Geschwindigkeitszuwachs profitieren, indem Probleme, die auf einem Quantencomputer schneller gelöst werden können, ausgelagert werden.

Damit eine Plattform, wie QUASP, vollständig QAaaS anbieten kann, muss die Plattform die Quanten-Services automatisch in der Cloud bereitstellen können. Dafür kann die Plattform die Anmeldedaten für die Cloud-Provider speichern. Dadurch ist es möglich, dass sich die Plattform über die Command-Line Interfaces (CLIs) der Provider anmeldet und den Quanten-Service automatisch in die Cloud hochlädt und alle benötigten Abhängigkeiten installiert. Die Abhängigkeiten für die Python-Umgebung sind in den Quanten-Services enthalten, jedoch können zusätzliche Programme benötigt werden, um diese Abhängigkeiten zu installieren.

Literaturverzeichnis

[AAA+19] H. Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110 (zitiert auf S. 36).

[ADK+04] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, O. Regev. *Adiabatic Quantum Computation is Equivalent to Standard Quantum Computation*. 2004. arXiv: quant-ph/0405098 [quant-ph] (zitiert auf S. 20).

[Azu] M. Azure. URL: https://azure.microsoft.com/de-de/ (zitiert auf S. 52).

[BIS+19] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, C. Blank, K. McKiernan, N. Killoran. „PennyLane: Automatic differentiation of hybrid quantum-classical computations“. In: *arXiv e-prints* (2019) (zitiert auf S. 38).

[BKM07] P. Bianco, R. Kotermanski, P. Merson. *Evaluating a Service-Oriented Architecture*. Techn. Ber. CMU/SEI-2007-TR-015. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2007. URL: http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=8443 (zitiert auf S. 27).

[CBSG17] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta. *Open Quantum Assembly Language*. 2017. arXiv: 1707.03429 [quant-ph] (zitiert auf S. 29, 36).

[CEP+18] P. J. Coles, S. Eidenbenz, S. Pakin, A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev, D. Gunter, S. Karra, N. Lemons, S. Lin, A. Lokhov, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'Malley, D. Oyen, L. Prasad, R. Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. Swart, M. Vuffray, J. Wendelberger, B. Yoon, R. Zamora, W. Zhu. *Quantum Algorithm Implementations for Beginners*. 2018. arXiv: 1804.03719 [cs.ET] (zitiert auf S. 26, 27).

[Cir] Cirq. *Cirq: A python library for NISQ circuits*. URL: https://cirq.readthedocs.io (zitiert auf S. 36).

[D-Wa] D-Wave. *Customers*. URL: https://www.dwavesys.com/our-company/customers (zitiert auf S. 17).

[D-Wb] D-Wave. *D-Wave's Ocean Software*. URL: https://ocean.dwavesys.com/ (zitiert auf S. 37).

[D-Wc] D-Wave. *The D-Wave 2000Q ™ System*. URL: https://www.dwavesys.com/d-wave-two-system (zitiert auf S. 32).

[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. 22).

[DiV00] D. P. DiVincenzo. „The Physical Implementation of Quantum Computation“. In: *Fortschritte der Physik* 48.9-11 (Sep. 2000), S. 771–783. ISSN: 1521-3978. DOI: 10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e. URL: http://dx.doi.org/10.1002/1521-3978(200009)48:9/11%3C771::AID-PROP771%3E3.0.CO;2-E (zitiert auf S. 19).

[Dja] Django. *The web framework for perfectionists with deadlines*. URL: https://www.djangoproject.com/ (zitiert auf S. 47).

[DMV01] W. van Dam, M. Mosca, U. Vazirani. „How powerful is adiabatic quantum computation?“ In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science* (2001). DOI: 10.1109/sfcs.2001.959902. URL: http://dx.doi.org/10.1109/SFCS.2001.959902 (zitiert auf S. 20).

[EH07] T. Ekman, G. Hedin. „The JastAdd system—modular extensible compiler construction“. In: *Science of Computer Programming* 69.1-3 (2007), S. 14–26 (zitiert auf S. 30).

[FGGS00] E. Farhi, J. Goldstone, S. Gutmann, M. Sipser. *Quantum Computation by Adiabatic Evolution*. 2000. arXiv: quant-ph/0001106 [quant-ph] (zitiert auf S. 20).

[Fie00] R. T. Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. AAI9980887. Diss. University of California, Irvine, 2000. ISBN: 0-599-87118-0 (zitiert auf S. 27, 28).

[Fina] D. Finke. *Qubit Quality*. URL: https://quantumcomputingreport.com/scorecards/qubit-quality/ (zitiert auf S. 32).

[Finb] D. Finke. *Tools*. URL: https://quantumcomputingreport.com/resources/tools/ (zitiert auf S. 31).

[Fin19] D. Finke. *Qubit Technology*. Sep. 2019. URL: https://quantumcomputingreport.com/scorecards/qubit-technology/ (zitiert auf S. 17).

[Fla] Flask. *web development, one drop at a time*. URL: https://flask.palletsprojects.com/en/1.1.x/ (zitiert auf S. 48).

[Fou] Q. O. S. Foundation. *List of Open Quantum Projects*. URL: https://qosf.org/project_list/ (zitiert auf S. 31).

[Gro96] L. K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph] (zitiert auf S. 17, 25).

[HM08] M. D. Hill, M. R. Marty. „Amdahl's Law in the Multicore Era“. In: *Computer* 41.7 (Juli 2008), S. 33–38. DOI: 10.1109/MC.2008.209 (zitiert auf S. 17).

[Hom15] M. Homeister. *Quantum Computing verstehen*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015. ISBN: 978-3-658-10454-2. DOI: 10.1007/978-3-658-10455-9 (zitiert auf S. 19, 22–26).

[IBM] IBM. *IBM Q Experience*. URL: https://quantum-computing.ibm.com/ (zitiert auf S. 17, 24, 30, 32).

[KIQ+19] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, C. Weedbrook. „Strawberry Fields: A Software Platform for Photonic Quantum Computing“. In: *Quantum* 3 (März 2019), S. 129. ISSN: 2521-327X. DOI: 10.22331/q-2019-03-11-129. URL: https://doi.org/10.22331/q-2019-03-11-129 (zitiert auf S. 32, 37).

[Kora] P. Korponaić. *Quantum Programming Studio*. URL: https://quantum-circuit.com/ (zitiert auf S. 29).

[Korb] P. Korponaić. *quantum-circuit*. URL: https://quantum-circuit.com/docs/quantum_circuit (zitiert auf S. 29).

[LaR19] R. LaRose. „Overview and Comparison of Gate Level Quantum Software Platforms“. In: *Quantum* 3 (März 2019), S. 130. ISSN: 2521-327X. DOI: 10.22331/q-2019-03-25-130. URL: https://doi.org/10.22331/q-2019-03-25-130 (zitiert auf S. 32, 37).

[LP17] V. Lahtinen, J. Pachos. „A Short Introduction to Topological Quantum Computation“. In: *SciPost Physics* 3.3 (Sep. 2017). ISSN: 2542-4653. DOI: 10.21468/scipostphys.3.3.021. URL: http://dx.doi.org/10.21468/SciPostPhys.3.3.021 (zitiert auf S. 20).

[MDL+18] A. McCaskey, E. Dumitrescu, D. Liakh, M. Chen, W. Feng, T. Humble. „A language and hardware independent approach to quantum-classical computing“. In: *SoftwareX* 7 (2018), S. 245–254. ISSN: 2352-7110. DOI: 10.1016/j.softx.2018.07.007. URL: http://www.sciencedirect.com/science/article/pii/S2352711018300700 (zitiert auf S. 39).

[MDLH18] A. McCaskey, E. Dumitrescu, D. Liakh, T. Humble. „Hybrid Programming for Near-Term Quantum Computing Systems“. In: *2018 IEEE International Conference on Rebooting Computing (ICRC)*. Nov. 2018, S. 1–12. DOI: 10.1109/ICRC.2018.8638598 (zitiert auf S. 39).

[Mel10] I. Melzer. *Service-orientierte Architekturen mit Web Services*. Hrsg. von I. Melzer. Bd. 4. Springer Spektrum, 2010. ISBN: 978-3-8274-2549-2. DOI: 10.1007/978-3-8274-2550-8 (zitiert auf S. 27).

[MG11] P. M. Mell, T. Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Techn. Ber. Gaithersburg, MD, United States: National Institute of Standards und Technology, 2011 (zitiert auf S. 17).

[Mic] Microsoft. *Microsoft Quantum Documentation*. URL: https://docs.microsoft.com/de-de/quantum/?view=qsharp-preview (zitiert auf S. 37).

[MMK+95] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, D. J. Wineland. „Demonstration of a Fundamental Quantum Logic Gate“. In: *Phys. Rev. Lett.* 75 (25 Dez. 1995), S. 4714–4717. DOI: 10.1103/PhysRevLett.75.4714. URL: https://link.aps.org/doi/10.1103/PhysRevLett.75.4714 (zitiert auf S. 20).

[PK11] C. A. Pérez-Delgado, P. Kok. „Quantum computers: Definition and implementations“. In: *Phys. Rev. A* 83 (1 Jan. 2011), S. 012303. DOI: 10.1103/PhysRevA.83.012303. URL: https://link.aps.org/doi/10.1103/PhysRevA.83.012303 (zitiert auf S. 19).

[Pro] ProjectQ. *Features of ProjectQ*. URL: https://projectq.ch/features/ (zitiert auf S. 38).

[RB01] R. Raussendorf, H. J. Briegel. „A One-Way Quantum Computer“. In: *Phys. Rev. Lett.* 86 (22 Mai 2001), S. 5188–5191. DOI: 10.1103/PhysRevLett.86.5188. URL: https://link.aps.org/doi/10.1103/PhysRevLett.86.5188 (zitiert auf S. 21).

[Riga] Rigetti. *https://pyquil.readthedocs.io/en/stable/*. URL: https://pyquil.readthedocs.io/en/stable/ (zitiert auf S. 36).

[Rigb] Rigetti. *QPU Specifications*. URL: https://www.rigetti.com/qpu (zitiert auf S. 32).

[Rod08] A. Rodriguez. *Restful web services: The basics*. 2008 (zitiert auf S. 27).

[SCZ17] R. S. Smith, M. J. Curtis, W. J. Zeng. *A Practical Quantum Instruction Set Architecture*. 2017. URL: https://arxiv.org/abs/1608.03355v2 (zitiert auf S. 36).

[SFN06] S. D. Sarma, M. Freedman, C. Nayak. „Topological quantum computation“. In: *Physics Today* 59.7 (2006), S. 32–38 (zitiert auf S. 20).

[Sho94] P. W. Shor. „Algorithms for quantum computation: discrete logarithms and factoring“. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. Nov. 1994, S. 124–134. DOI: 10.1109/SFCS.1994.365700 (zitiert auf S. 17, 26).

[SHT18] D. S. Steiger, T. Häner, M. Troyer. „ProjectQ: an open source software framework for quantum computing“. In: *Quantum* 2 (Jan. 2018), S. 49. ISSN: 2521-327X. DOI: 10.22331/q-2018-01-31-49. URL: https://doi.org/10.22331/q-2018-01-31-49 (zitiert auf S. 29, 38).

[Siu05] M. S. Siu. „From quantum circuits to adiabatic algorithms“. In: *Physical Review A* 71.6 (Juni 2005). ISSN: 1094-1622. DOI: 10.1103/physreva.71.062314. URL: http://dx.doi.org/10.1103/PhysRevA.71.062314 (zitiert auf S. 41).

[SS14] H. Singh, A. Sachdev. „The Quantum way of Cloud Computing“. In: *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*. Feb. 2014, S. 397–400. DOI: 10.1109/ICROIT.2014.6798362 (zitiert auf S. 17).

[VB15] B. Varanasi, S. Belida. „HATEOAS“. In: *Spring REST*. Berkeley, CA: Apress, 2015, S. 165–174. ISBN: 978-1-4842-0823-6. DOI: 10.1007/978-1-4842-0823-6_10. URL: https://doi.org/10.1007/978-1-4842-0823-6_10 (zitiert auf S. 28).

[WBL+15] J. Wettinger, U. Breitenbücher, F. Leymann et al. „Any2API-Automated APIfication“. In: *Proceedings of the 5th International Conference on Cloud Computing and Services Science. SciTePress*. 2015 (zitiert auf S. 30).

[WRR+05] P. Walther, K. J. Resch, T. Rudolph, E. Schenck, H. Weinfurter, V. Vedral, M. Aspelmeyer, A. Zeilinger. „Experimental one-way quantum computing“. In: *Nature* 434.7030 (März 2005), S. 169–176. ISSN: 1476-4687. DOI: 10.1038/nature03347. URL: http://dx.doi.org/10.1038/nature03347 (zitiert auf S. 21).

[WZY05] X. R. Wang, Y. S. Zheng, S. Yin. „Spin relaxation and decoherence of two-level systems“. In: *Physical Review B* 72.12 (Sep. 2005). ISSN: 1550-235X. DOI: 10.1103/physrevb.72.121303. URL: http://dx.doi.org/10.1103/physrevb.72.121303 (zitiert auf S. 32).

[XAC] XACC. *Welcome to XACC's documentation!* URL: https://xacc.readthedocs.io/en/latest/ (zitiert auf S. 39).

[YHW17] H. Yu, Y. Huang, B. Wu. *Exact Equivalence between Quantum Adiabatic Algorithm and Quantum Circuit Algorithm*. 2017. arXiv: 1706.07646 [quant-ph] (zitiert auf S. 20).

Alle URLs wurden zuletzt am 22. 10. 2019 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift