Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Mustererstellung für Quantenalgorithmen in einem Musterrepository

Martin Beisel

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	Marie Salm M.Sc., Manuela Weigold, M.Sc., Karoline Wild, M.Sc.
Beginn am:	4. Februar 2020
Beendet am:	29. September 2020

Kurzfassung

Muster werden in unterschiedlichsten Domänen zur Dokumentation von häufig wiederkehrenden Problemen verwendet. Zumeist arbeiten mehrere Musterautoren kollaborativ an dem Identifikations- und Erstellungsprozess einer Mustersprache. Die Dokumentation der Muster erfolgt entweder klassisch in einem Buch oder Paper oder in einem Musterrepository. Ein Repository bietet den Vorteil, dass es einfach angepasst und erweitert werden kann. Mit der steigenden Relevanz des Quantencomputings wurde es immer wichtiger, dass Entwickler Quantenalgorithmen effizient erstellen können. Um dies zu erreichen wurden häufig wiederkehrende Probleme aufgefasst und in den ersten Quantencomputing-Mustern dokumentiert. Da sich die Quantencomputing-Domäne momentan sehr schnell entwickelt, sollen die Muster erweitert und beim Erlangen neuer Erkenntnisse angepasst werden. Im Rahmen dieser Masterarbeit wurde analysiert, welche spezifischen Anforderungen Quantencomputing-Muster an ein Musterrepository haben. Insbesondere das Darstellen, Bearbeiten und Diskutieren von Quantenartefakten, wie mathematischen Formeln, Quantenschaltungen und konkreten Implementierungen, wurde als essentiell identifiziert. Auf Basis dieser Anforderungen und der allgemeinen Richtlinien zum Mustererstellungsprozess wurde ein Konzept für ein Quantencomputing-Musterrepository entwickelt, welches im Pattern Atlas als Prototyp umgesetzt wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

Mit den in den letzten Jahren erreichten Fortschritten in der Entwicklung von Quantencomputern steigt die Relevanz des Quantencomputings rasant [Gas18]. Quantencomputer haben das Potential bestimmte hochkomplexe Berechnungen effizient lösen. So soll es in der Zukunft möglich sein, Rechnungen und Simulationen durchzuführen, welche mit einem klassischen Computer unmöglich zu lösen wären [Gid19] [Kan98]. Dies ist möglich, indem man die Gesetze der Quantenmechanik ausnutzt. Jedoch erfordert die Entwicklung entsprechender Quantenalgorithmen ein tiefes mathematisches und quantenmechanisches Verständnis. Gerade für Entwickler mit wenig Erfahrung im Quantencomputing, stellt die Entwicklung solcher Algorithmen eine große Herausforderung dar. Um Entwickler bei der Entwicklung neuer Algorithmen zu unterstützen, hat Leymann [Ley19] erste Quantencomputing-Muster identifiziert und zu einer Mustersprache vereinigt. Muster fassen häufig wiederkehrende Probleme auf und beschreiben deren Lösungen auf eine abstrakte Weise. Die Quantencomputing-Mustersprache soll Entwicklern, welche einen Quantenalgorithmus entwickeln, dabei helfen die Konzepte des Quantencomputings besser zu verstehen und darin wiederkehrende Probleme erkennen und lösen zu können.

Um Lösungen für wiederkehrende Probleme mit anderen Programmierern teilen zu können, wurde bereits ein Musterrepository entwickelt, in dem unterschiedlichste Mustersprachen erfasst und zur Verfügung gestellt werden [IAA20]. In diesem sollen neben den bereits existierenden Mustersprachen wie Cloud Computing [FLR+14] zukünftig auch die Quantencomputing-Muster eingepflegt und kooperativ weiterentwickelt werden. Die Umsetzung des Mustererstellungsprozesses im Musterrepository ist stark an [FBBL15] orientiert. Muster werden in einem kollaborativen, iterativen Prozess kontinuierlich hinsichtlich ihres Inhaltes und dessen Darstellung erweitert und optimiert.

Quantenalgorithmen können mit mathematischen Formeln beschrieben werden. Um diese übersichtlicher darzustellen, werden die Formeln häufig in äquivalente Quantengatter und Quantenschaltkreise umgewandelt. Die Darstellung solcher Formeln und deren äquivalenten grafischen Darstellungen stellt einen wichtigen Bestandteil der Quantencomputing-Muster dar. Die grafische Darstellung dieser Formeln und Gatter erfolgt momentan in den meisten Fällen mit LaTeX [CTA20] oder unterschiedlichen Modellierungstools. LaTeX ist de facto der Standard für wissenschaftliche Arbeiten und basiert auf dem Textsatzsystem TeX, dessen Verwendung durch Makros vereinfacht wird. Da es keine Unterstützung der Tools zur Darstellung von Quantenalgorithmen in jetzigen Musterrepositories gibt widerspricht diese der Prämisse, dass Muster im Musterrepository einfach, einheitlich und kollaborativ entwickelt werden können. Demzufolge müssen Anpassungen vorgenommen werden, sodass mathematische Formeln und Gatterdarstellungen nicht nur einheitlich und übersichtlich im Musterrepository dargestellt, sondern auch einfach und schnell zu kommentieren und zu bearbeiten sind. In dieser Arbeit werden die spezifischen Anforderungen an das Musterrepository analysiert, ein Konzept erarbeitet und anschließend in einem Prototypen umgesetzt.

Zuerst werden in Kapitel 2 die zum Verständnis notwendigen Grundlagen und Definitionen eingeführt. Anschließend werden in Kapitel 3 Arbeiten und Projekte vorgestellt, welche ähnliche Konzepte wie diese Arbeit behandeln. In Kapitel 4 wird genauer auf Leymanns Quantencomputing-Muster und die Vision für die Umsetzung dieser in einem Musterrepository eingegangen. In Kapitel 5 werden die Anforderung an das Musterrepository und die zur Umsetzung notwendigen Tools analysiert. Darauf folgenden wird in Kapitel 6 das Konzept zur Verbesserung des Mustererstellungsprozesses für Quantencomputing-Muster in einem Musterrepository vorgestellt. In Kapitel 7 wird der Umsetzungsprozess des Prototypen dokumentiert und erläutert. Anschließend wird der Prototyp in Kapitel 8 anhand eines Beispielszenarios ausgewertet. Im letzten Kapitel werden die Ergebnisse dieser Masterarbeit zusammengefasst und mögliche Anknüpfungspunkte für zukünftige Arbeiten vorgestellt.

2 Grundlagen

In diesem Kapitel werden die zum Verständnis dieser Arbeit notwendigen Grundlagen und Begriffe erläutert. Zuerst wird die Funktionsweise von Mustern und Mustersprachen erklärt. Darauf folgend wird deren iterativer Erstellungsprozess vorgestellt und kurz auf die Integration von Mustern in einem Musterrepository eingegangen. Als nächstes werden die Grundlagen des Quantencomputings erläutert. Darauf folgend werden die Quantencomputing-Muster vorgestellt. Im letzten Abschnitt wird die Quantum Assembly Language (QASM) vorgestellt.

2.1 Muster & Mustersprachen

Muster fassen häufig wiederkehrende Probleme auf und beschreiben deren Lösungen auf eine abstrakte Weise. Muster werden als strukturierte Textdokumente festgehalten. Sie erläutern, wie ein Problem gelöst werden soll, welche Ideen dahinter stecken und warum der vorgestellte Lösungsweg anderen Lösungswegen vorzuziehen ist. Die abstrakten Lösung helfen also das zugrunde liegende Problem zu verstehen, um so eine konkrete Lösung zu finden. Ein Muster erlaubt eine nahezu unbegrenzte Anzahl an konkreten Lösungen für das vorliegende Problem [AIS77]. Der Grundbegriff der Muster hat seine Wurzeln in der Städtearchitektur [AIS77], wo Muster zur Gebäude- und Stadtplanung entwickelt wurden. Schon bald wurde das Prinzip der Muster von anderen Domänen, insbesondere der Softwarearchitektur aufgegriffen [GHJV94] [FLR+14] [Ley19]. Eine wichtige Charakteristik von Mustern ist, dass sie mit anderen Mustern kombiniert werden können, welche dann eine Mustersprache bilden [Ley19]. Typischerweise bauen Muster gezielt aufeinander auf, sodass diese gemeinsam ein größeres Problem lösen [Ley19]. Eine Mustersprache umfasst eine Menge von einheitlich formatieren Mustern, welche miteinander verknüpft sind. So haben beispielsweise alle Muster der Quantencomputing-Mustersprache die folgende Struktur: Name, Zweck, Icon, Problem, Kontext, Lösung, Known Uses, Next. Die Verknüpfungen ermöglichen eine Navigation zwischen den verschiedenen Mustern und geben einen Überblick darüber, welche Muster häufig in Kombination miteinander verwendet werden und womöglich bei der Umsetzung eines anderen Musters helfen können. Da die der Erstellung einer Mustersprache ein breite Wissensbasis und ein gut strukturiertes Vorgehen erfordern, empfehlen Fehling et al. [FBBL15] einen Prozess, der aus drei Phasen besteht: einer Identifikations-, Erstellungs- und Anwendungsphase. Dieser wird in Abschnitt 2.2 erläutert.

2.2 Musteridentifikation, Erstellung & Anwendung

Der in Abbildung 2.1 dargestellte iterative Prozess, stellt die von Fehling et al. [FBBL15] identifizierten Phasen bei der Erstellung einer Mustersprache dar. Fehling et al. [FBBL15] haben über mehrere Jahre hinweg Muster in unterschiedlichen Domänen recherchiert, identifiziert, verfasst

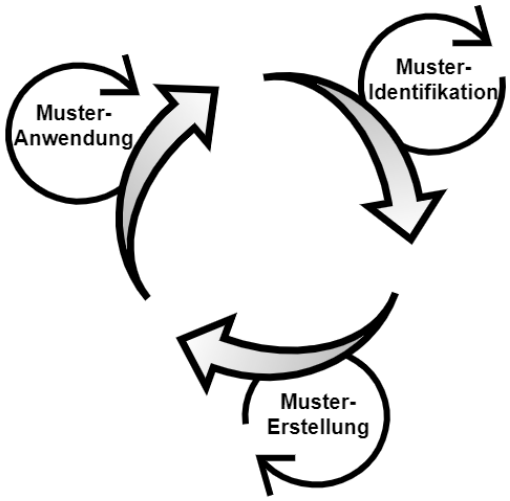


Abbildung 2.1: Musteridentifikations-, Erstellungs- & Anwendungsprozess angelehnt an Fehling et al. [FBBL15]

und angewendet. Bei der Analyse dieser Projekte konnten die angewendeten Praktiken in eine Musteridentifikations-, Mustererstellungs- und Musteranwendungsphase aufgeteilt werden, welche domänenunabhängig gültig sind. Sowohl der gesamte Prozess, als auch die einzelnen Phasen werden iterativ ausgeführt, um die bereits erstellten Ergebnisse zu diskutieren und kontinuierlich anzupassen und zu verbessern. Während der Musteridentifikations- und Mustererstellungsphase werden neue Muster in der ausgewählten Domäne identifiziert und anschließend in einem kollaborativen Prozess erstellt. In der Musteranwendungsphase werden die zuvor erstellten Muster schließlich für einen konkreten Anwendungsfall verfeinert. Dabei kann es sich beispielsweise um die spezifische Infrastruktur eines Unternehmens handeln [FBBL15]. Da der Fokus dieser Arbeit auf den ersten beiden Phasen liegt, werden nur diese im Folgenden genauer erläutert.

2.2.1 Musteridentifikation

Während der in Abbildung 2.2 dargestellten Musteridentifikationsphase werden relevante Informationen über die ausgewählte Domäne gesammelt und strukturiert. Das Hauptziele der Identifikationsphase ist es relevante Muster einer Domäne zu identifizieren und ein einheitliches Format für diese zu definieren. Da eine einheitliche Struktur essentiell für das Verständnis der Muster ist, sollte vor allem bei größeren Teams ein hoher Wert auf die Identifikationsphase gelegt werden.

1. **Domänendefinition:** Wesentliche Informationen und allgemein anerkannte Definitionen der Domäne werden zusammengetragen und dienen als allgemeine Informatiosgrundlage für alle beteiligten Musterautoren.
2. **Abdeckungsabwägung:** Die Musterautoren identifizieren die relevanten Themen der Domäne und legen die zu verwendenden Informationsquellen fest.
3. **Informationsformat:** Die Musterautoren einigen sich auf ein Format zur Darstellung der gesammelten Informationen.

4. **Informationssammlung:** Die ausgewählten Informationsquellen werden nach Lösungen durchsucht, welche dann im definierten Informationsformat gesammelt werden.
5. **Informationsreview:** Die Domänenstruktur wird verfeinert, sodass kleinere, handhabbare Lösungsmengen entstehen, welche miteinander verglichen und auf Ähnlichkeiten überprüft werden können.

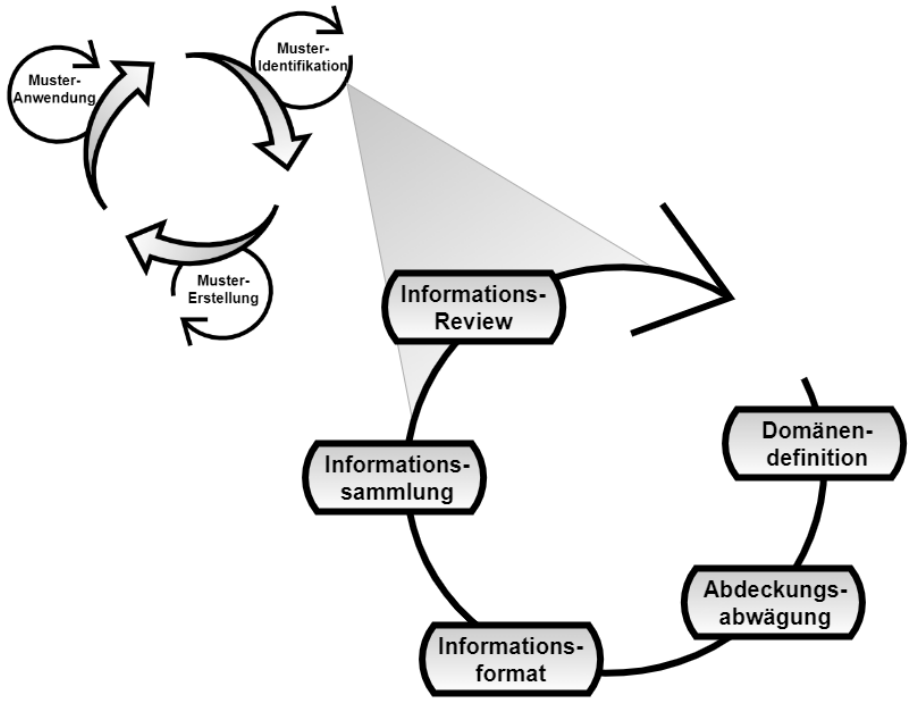


Abbildung 2.2: Musteridentifikationsphase angelehnt an Fehling et al. [FBBL15]

2.2.2 Mustererstellung

Während der in Abbildung 2.3 dargestellten Mustererstellungsphase werden, aus den in der Musteridentifikationsphase gefundenen Lösungsähnlichkeiten, Muster verfasst.

1. **Mustersprachendesign:** Das konkrete Musterformat wird basierend auf den allgemein anerkannten Musterformaten und den domänenspezfischen Anforderungen definiert. Dies beinhaltet auch die Semantik der einzelnen Abschnitte, beispielsweise, welche Informationen in welcher Länge in einem Abschnitt enthalten sind und die Semantik der Wechselbeziehungen zwischen den Mustern.
2. **Basisdefintionen:** Standardisierung und Verfeinerung der in der Musteridentifikationsphase festgelegten Darstellungsformate.
3. **Definition der Darstellungssprache:** Viele Muster verwenden Skizzen, um abstrakte Lösungen grafisch zu beschreiben. Um eine einheitliche Darstellung zu garantieren, werden Richtlinien und Standards zu Grafikdarstellung festgelegt.

4. **Muster verfassen:** Beim Verfassen der Muster liegt eine der Hauptschwierigkeiten darin, das richtige Abstraktionslevel zu wählen, sodass die Lösung weder zu wenige Informationen enthält noch zu konkret ist, um sie auf neue Vorkommen des Problems zu übertragen. Dazu soll das Musterdokument iterativ von Dritten, welche Experten oder Novizen in der Domäne sind, beurteilt werden.
5. **Überarbeitung der Mustersprache:** Während dieser Phase sollen die Wechselbeziehungen zwischen den Mustern erneut überprüft und aktualisiert werden. Insbesondere am Anfang verfasste Muster tendieren dazu weniger Verknüpfungen zu haben oder fehlende Verknüpfungen bei bidirektionalen Beziehungen zu beinhalten.

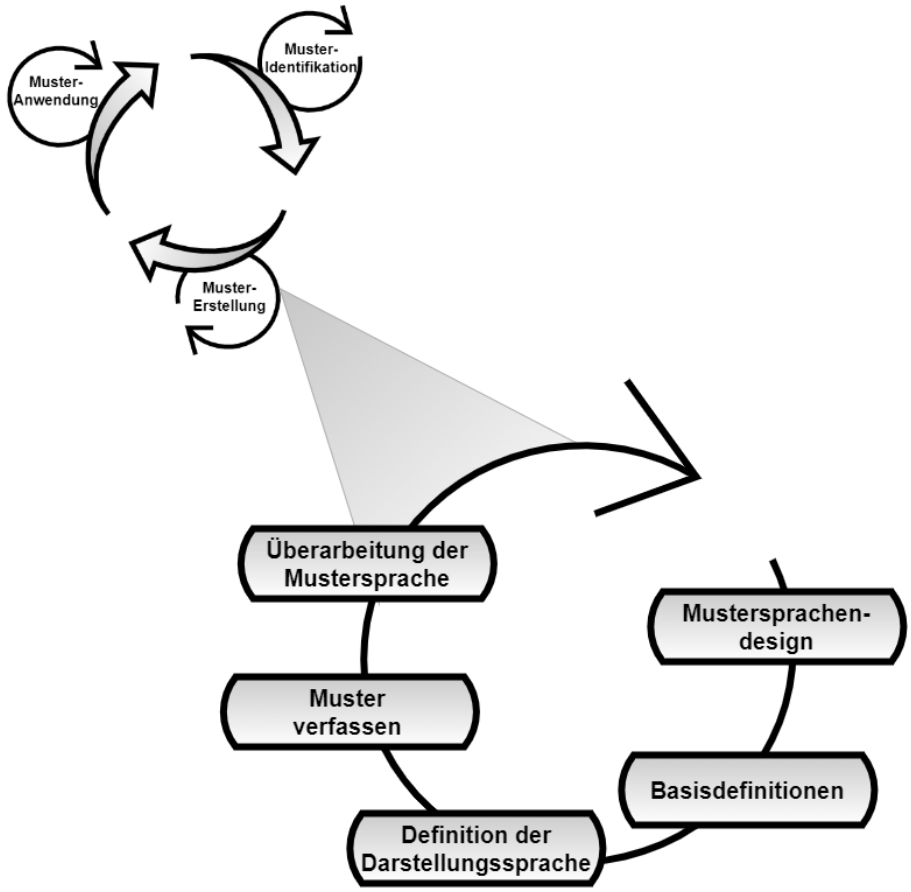


Abbildung 2.3: Mustererstellungsphase angelehnt an Fehling et al. [FBBL15]

2.3 Musterrepository

Ein Musterrepository ist eine auf Muster spezialisierte Datenbank, in welcher Muster und deren Verknüpfungen gespeichert werden können. In einem Musterrepository kann gezielt nach Mustern gesucht werden, um deren Inhalt einzusehen. Des Weiteren kann zwischen verknüpften Mustern

navigiert werden [Ley19]. Da Muster kollaborativ erstellt werden, verfügen Musterrepositories typischerweise über ein Frontend, in welchem die dokumentierten Muster übersichtlich dargestellt und bearbeitet werden können.

2.4 Quantencomputer

Mit den in den letzten Jahren erzielten Fortschritten in der Entwicklung von Quantencomputern wurden große Schritte zum Erreichen neuer Durchbrüche im Bereich der Forschung und Wissenschaft getan [IBM20c]. Quantencomputer sollen jedoch nicht den klassischen Computer ersetzten, sondern dienen als Ergänzung für die Berechnung hochkomplexer Probleme, die selbst ein moderner Supercomputer unmöglich, in realistischer Zeit lösen kann [IBM20c] [Pre18]. Beispielsweise sollen Quantencomputer zur Modellierung modularer Interaktionen verwendet werden, um bessere Arzneimittel und andere chemische Produkte zu entwickeln [Jac17]. Um eine Chance zu haben diese Art von Probleme zu lösen, muss eine neue Art von Computern entwickelt werden. Quantencomputer machen sich Eigenschaften der Quantenmechanik zu Nutze. Sie nutzen das Phänomen der Superposition und der Verschränkung. Für einen Quantencomputer bedeutet das Prinzip der Superposition, dass er nicht nur auf die Zustände 0 und 1 beschränkt ist, sondern auch in einer Kombination beider Zustände sein kann [CEP+18] [KLM+07]. Der Zustand eines Quantencomputers wird nicht, wie in klassischen Computern, in *Bits*, sondern in *Qubits* gespeichert [CEP+18]. Berechnungen auf Quantencomputern werden mittels Quantenalgorithmen durchgeführt. Diese Arbeit fokussiert sich auf gatterbasierte Quantenalgorithmen. Analog zu den im klassischen Computer auf Bits angewendeten Gattern, wie NOT, AND, OR, XOR, etc. existieren Quantengatter, welche auf Qubits angewendet werden. Auf einem Quantencomputer werden Quantengatter und Messungen ausgeführt. Mit den Messungen werden die Ergebnisse der Berechnung bestimmt [Roe18]. Ein Quantensystem besteht aus mehreren Qubits und der Zustand des Quantensystems wird immer durch einen Vektor in einem komplexen Vektorraum beschrieben [CEP+18]. Quantenalgorithmen können immer als Transformation, welche in diesem Vektorraum ausgeführt wird, ausgedrückt werden [CEP+18]. Transformationen, welche zum Bau von komplizierteren Transformationen benutzt werden, werden als Gatter bezeichnet [CEP+18]. Um Quantenalgorithmen übersichtlicher darzustellen, gibt es Abkürzungen und Buchstabenbezeichnungen für die wichtigsten Gatter [CEP+18]. So wird beispielsweise das Hadamard-Gatter $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$ durch das Kürzel H identifiziert. Mehrere Quantengatter können dann zu einem Quantenschaltkreis kombiniert werden [CEP+18]. Um Quantenschaltkreise zu modellieren, wird häufig auf Tools zurückgegriffen. Diese ermöglichen es Quantengatter einfach mittels Drag & Drop zu einem Quantenschaltkreis zu kombinieren. In der Regel kann auf eine Auswahl an Gattern zugegriffen werden, welche auf die einzelnen Qubits gezogen werden können. Qubits werden im Allgemeinen durch horizontale Linien repräsentiert. Ein Beispiel für die Oberfläche eines solchen Modellierungstools ist in Abbildung 5.2 zu sehen.

2.5 Muster für Quantencomputing

Da Quantencomputer eine völlig neue Herangehensweise an die Problemlösung voraussetzen, unterscheiden sich Quantenalgorithmen deutlich von klassischen Algorithmen. Das Programmieren von Quantencomputern ist selbst für erfahrene Softwareentwickler eine völlig neue Herausfor-

derung. Deshalb entwickelt Leymann [Ley19] eine Mustersprache für das Programmieren von Quantencomputern. In Leymanns Quantencomputing-Mustern werden bereits erkannte „Tricks“ aus Fachbüchern zusammengefasst und in klar strukturierten Mustern dokumentiert. Ziel ist es eine übersichtliche Sammlung zur Lösung von Problemen im Kontext des Quantencomputings zu erstellen [Ley19].

2.6 QASM

Quantum Assembly Language (QASM) ist eine einfache Textsprache, welche generische Quantenschaltkreise beschreibt [CBSG17]. Sie tauchte erstmals 2005 in [Cro05] in einem Softwarepaket auf [KGA+18]. Mit QASM können Quantencomputing-Programme dargestellt werden, deren Parameter vollständig definiert sind [CBSG17]. Es gibt eine Vielzahl unterschiedlicher auf bestimmte Anwendungen optimierte QASM-Dialekte [KGA+18]. In dieser Arbeit wird der Industriestandard [SDC+20] OpenQASM [IBM20a] verwendet. OpenQASM ist eine einfache low-level Programmiersprache. Programme in OpenQASM bestehen aus einer Sequenz von Deklarationen und Kommandos [AG20]. Es können sowohl klassische als auch Quantenregister verwendet werden. Die vollständige OpenQASM Dokumentation kann in [CBSG17] gefunden werden.

2.7 SVG

Scalable Vector Graphics (SVG) ist eine XML-basierte Markup-Sprache zur Beschreibung von zwei-dimensionalen Vektorgrafiken [Moz20b]. SVG-Bilder und deren Verhalten sind in XML-Textdateien definiert und können als solche durchsucht, indexiert, geskriptet oder komprimiert werden [Moz20b]. Dies bietet außerdem den Vorteil, dass die Bilder mit jedem Texteditor bearbeitet werden können. Zum Bearbeiten ist also kein grafischer Editor nötig. Ein weiterer Vorteil von SVG gegenüber klassischen Rastergrafikformaten wie PNG oder JPG ist, dass die Grafiken ohne Qualitätsverlust skalierbar sind [Moz20b]. Rastergrafiken beschreiben, welche Farben die einzelnen Pixel eines fest definierten Rasters haben. SVG-Darstellungen werden auf Basis eines internen Koordinatensystem definiert [Bel17]. Dieses wird beim Rendern entsprechend der vom Nutzer festgelegten Größe skaliert [Moz20b].

3 Related Work

Als Grundlage dieser Arbeit wurden nicht nur die Quantencomputing-Muster [Ley19] und der Pattern Atlas [IAA20] verwendet, sondern auch auf die Erfahrungen und Konzepte existierender Arbeiten, Projekte und Repositories zurückgegriffen. Zwar gibt es noch kein veröffentlichtes Quantencomputing-Musterrepository, doch können Konzepte und Erfahrungen, besonders im Hinblick auf den kollaborativen Erstellungsprozess von Mustern, domänenübergreifend angewendet werden.

Fehling et al. [FBFL14] stellen das Tool *PatternPedia* zum kollaborativen Dokumentieren existierender Lösungen und zur Verwaltung, der aus diesen Lösungen extrahierten Mustern vor. Fehling et al. [FBFL14] haben anhand der Analyse des Mustererstellungsprozesses ein erweiterbares Muster-Metamodell erstellt. Das Muster-Metamodell ist in UML modelliert und als Beispiel für die allgemeine Erweiterbarkeit wurden die Cloudcomputing-Muster von Fehling et al. [FLR+14] und Muster zur Erstellung von Kostümen in das Metamodell integriert. PatternPedia basiert auf diesem Metamodell und dient als gutes Beispiel für ein Musterrepository, welches sowohl Lösungen, als auch den Identifizierungs- und Erstellungsprozess unterstützt.

Köppe et al. [KISV16] untersuchen eine Vielzahl von online verwendbaren Musterrepositories auf ihre Unterstützung des Musterlebenszyklus. Das Ziel der Analyse ist es herauszufinden, wieso trotz der gegebenen Verfügbarkeit an Musterrepositories die Popularität sehr gering ist. Als Hauptgründe werden die kleine Community von Stakeholder und die mangelnde vollständige Unterstützung des Musterlebenszyklus gegeben. Köppe et al. [KISV16] sind der Meinung, dass der letzterer Grund einen Einfluss auf die Größe der Community hat. [KISV16] fokussiert sich auf die Identifikation der Stakeholderbedürfnisse und die Umsetzung dieser.

Reiners et al. [RFJZ15] analysieren den Erstellungsprozess von existierenden Mustersprachen mit dem Ziel Muster als *lingua franca* zwischen Stakeholdern einer Domäne zu etablieren. Forschungsprojekte haben häufig 50-100 Beteiligte, welche unterschiedliche Erfahrungen, Spezialisierungen und Wissensstände haben und verschiedene Technologien und Prozessabläufe gewohnt sind. Dadurch entsteht ein sehr hoher Kommunikationsaufwand. Um dieses Problem zu lösen, sollen Muster als Mikro-Dokumentationen dienen, welche von den Stakeholdern eines Forschungsprojekts erstellt und unter ihnen einfach geteilt werden können. In [RFJZ15] werden Musterrepositories analysiert und Anforderungen entsprechend der Mängel identifiziert. Die Anforderungen fokussieren sich insbesondere auf den kollaborativen Bearbeitungsprozess von Mustern.

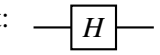
Deng et al. [DKT05] analysieren eine Vielzahl von Tools zum Verwalten von User-Interface-Mustersprachen. Als Hauptprobleme für die Umsetzung der UI-Muster werden ein fehlendes Standardformat, fehlende Versionierungsoptionen und eine mangelhafte Einbeziehung, der auf ein Muster wirkenden Einflüsse, identifiziert. Des Weiteren wird auf die Wichtigkeit eines flexiblen Musterformats zur Mustererstellung eingegangen.

Inventado und Scupelli [IS15] stellen ein Konzept zum kollaborativen Arbeiten während des gesamten Musterlebenszyklus vor. Die Arbeit stellt das *Data-driven design pattern production* (3D2P) Konzept zur Erstellung von Online-Lernplattform-Mustern vor. Die Besonderheit des 3D2P-Konzeptes ist die stark ausgeprägte Einbindung von Feedback und Kontrollverfahren, mit welchen die Qualität der Muster sichergestellt werden sollen. Inventado und Scupelli [IS15] wollen mit den gesammelten Feedback-Daten einen kontinuierlichen Verfeinerungsprozess weiter unterstützen.

Quantum Computing Stack Exchange [Sta20] ist eine jedem zugänglichen Online-Plattform, auf welcher Quantencomputing-Interessierte Fragen stellen und beantworten können. Die Plattform hat mit über 10.000 registrierten Nutzern eine breite Nutzerbasis, auf Basis welcher beispielsweise typische Darstellungsmethoden erkannt werden können. Fragen und Antworten folgen keinem bestimmten Format und Verknüpfungen zwischen Fragen müssen manuell von Nutzern als Link in eine Antwort integriert werden. Fragen können mit Tags annotiert und mittels dieser gefunden werden.

4 Vision

In [Ley19] hat Leymann das Fundament für eine umfassende Quantencomputing-Mustersprache gelegt. Momentan beinhaltet die Quantencomputing-Mustersprache nur zehn Muster, die beim Erstellen von Quantenalgorithmen beachtet werden sollen. So erläutert Leymann [Ley19] im ersten Muster, wie das zu manipulierende Quantenregister am Anfang eines Quantenalgorithmus initialisiert werden muss. In den darauf folgenden Mustern wird auf die Wichtigkeit und Anwendung der Prinzipien der Superposition und der Quantenverschränkung, sowie weitere Methoden zur Implementierung von Quantenalgorithmen eingegangen. Leymann [Ley19] legt viel Wert auf den logischen Aufbau der Muster. Dies führt dazu, dass die Leser die Muster direkt verknüpfen können und ein besseres Verständnis des Themas erlangen.

Allgemein beschreiben Muster abstrakte Lösungen, welche unabhängig von konkreten Umsetzungen sind. Dies hat den Vorteil, dass die Muster beispielsweise programmiersprachenunabhängig sind. So wird allgemein zwischen Mustersprachen und Lösungssprachen unterschieden. Der Unterschied liegt darin, dass eine Lösungssprache konkrete Lösungsartefakte für die abstrakten Muster einer Mustersprache beinhaltet [FL17]. Im Falle der Quantencomputing-Muster verschwimmt die klare Trennung zwischen Muster- und Lösungssprache. So enthalten das „Uniform Superposition“, „Creating Entanglement“, „Function Table“ und „Phase Shift“ Muster konkrete mathematische Formeln zur Beschreibung von Quantenzuständen. Beispielsweise kann eine „Uniform Superposition“ durch die Initialisierung des Quantenregisters als Einheitsvektor $|0\ldots0\rangle$ und die anschließende Anwendung der Hadamard Transformation erreicht werden. Dies wird mathematisch dann wie folgt dargestellt: $H^{\otimes n}(|0\rangle^{\otimes n}) = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^{n-1}} |x\rangle$ [Ley19]. Alle Formeln zur Beschreibung von Quantenzuständen können auch als äquivalentes Gatter dargestellt werden. Ein klassischer Computer wendet Gatter, wie NOT, AND, OR, etc. auf Bits an, analog werden Gatter im Quantencomputing auf Qubits angewendet. Ein Hadamard-Gatter wird wie folgt dargestellt:  [ZLC00]. Somit enthalten die Muster in [Ley19] Teile konkreter Lösungen.

Des Weiteren erklärt Leymann, dass die Muster möglicherweise durch konkrete Implementierungen der Quantencomputing-Muster in gängigen Formaten wie QASM ergänzt werden sollen [Ley19]. So wäre es möglich die Muster zu verfeinern und direkt anzuwenden. Außerdem soll die in [Ley19] dokumentierte Mustersprache im Musterrepository *Pattern Atlas* [IAA20] veröffentlicht werden.

Der Pattern Atlas unterstützt zwar den Mustererstellungsprozess, ist jedoch nicht für die Darstellung von mathematischen Formeln und Quantencomputing-Artefakten, wie Quantenschaltkreisen optimiert. In Kapitel 5 wird erörtert, welche Darstellungsformen und Annotationen für den Mustererstellungsprozess der Quantencomputing-Muster am besten geeignet sind und welche Regeln und Einschränkungen festgelegt werden müssen.

5 Analyse der Quantencomputing spezifischen Anforderungen

In diesem Kapitel werden die speziellen Quantencomputing spezifischen Anforderungen analysiert. Des Weiteren wird ausgewertet, welche Tools zur Einbindung von Quantencomputing-Implementierungen und Gatterdarstellungen verwendet werden.

5.1 Abgeleitete Anforderungen zur Dokumentation von Quantenalgorithmen in Mustern

Da sich die Quantencomputing-Muster, wie in Kapitel 4 beschrieben, von den bisher im Pattern Atlas dokumentierten Mustern unterscheiden, muss überprüft werden, ob der in Abschnitt 2.2 vorgestellte Mustererstellungsprozess für Quantencomputing-Muster im Pattern Atlas möglich ist und ob Anpassungen nötig sind. Gemäß [FBBL15] müssen alle in Abbildung 5.1 dargestellten Schritte unterstützt werden.

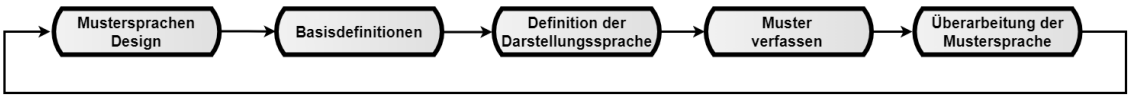


Abbildung 5.1: Mustererstellungsphase angelehnt an Fehling et al. [FBBL15]

Mustersprachen Design: Das von Leymann [Ley19] gewählte Musterformat wird im Pattern Atlas vollständig unterstützt. Es kann ein Mustername festgelegt werden, auf welchen ein Beschreibung des Zwecks des Musters und ein Icon folgt. Anschließend wird das in dem Muster gelöste Problem zusammengefasst. Im Kontext Abschnitt wird auf die Situation eingegangen, welche zu dem Problem geführt hat. Der nächste und wichtigste Abschnitt ist die Lösung des Problems. Im darauf folgenden Abschnitt werden bekannte Algorithmen gelistet, welche das Muster anwenden. Zuletzt werden mit dem behandelten Muster verwandte Muster referenziert. Diese Referenzen sollen als direkt navigierbare Verknüpfungen zwischen den einzelnen Mustern umgesetzt werden.

Basisdefinitionen: Da mathematische Formeln ein Hauptbestandteil der Quantencomputing-Muster sind, müssen diese einfach und einheitlich dargestellt werden können. Dies gilt auch für die Darstellungen von Quantenschaltkreisen und Quantengattern. Des Weiteren soll die Integration von Implementierungen in gängigen Sprachen, wie QASM berücksichtigt werden. Da sowohl die mathematischen Formeln, als auch die Quantenschaltkreise, die in das Musterrepository aufgenommen werden sollen, mit hoher Wahrscheinlichkeit Teil einer Veröffentlichung oder einer bestehenden

Implementierung sind, soll die Übernahme existierender Artefakte mit möglichst geringem Aufwand verbunden sein. LaTeX ist de facto der Standard zur Darstellung von mathematischen Formeln in wissenschaftlichen Dokumenten und somit allgemein bekannt [CTA20] [Gau07]. Die Darstellung von Quantenschaltkreisen in LaTeX-Dokumenten erfolgt zumeist [Deb19] mit Qcircuit [Bry20] oder Quantikz [Qua19]. Um die Darstellung von implementierten Quantenalgorithmen zu ermöglichen, soll es möglich sein den Quellcode zu integrieren, welcher dann als Quantenschaltkreis dargestellt wird. Momentan stellt IBM Q Experience mit tausenden Nutzern und hunderten Zitierungen eine der populärsten Quantenplattformen dar [GGA19]. So wurden während einer viertägigen globalen Quanatencomputing-Challenge durchschnittlich über eine Milliarde Quantenschaltkreise am Tag ausgeführt [GCBA20]. Die von IBM bereitgestellten Quantencomputer können mittels Qiskit [IBM20b] benutzt werden, welches auf der Quantum Assembly Sprache OpenQASM [IBM20a] basiert. Da OpenQASM de facto der Industriestandard ist [SDC+20], wird im Pattern Atlas der Fokus auf der Darstellung von OpenQASM liegen. Des Weiteren wäre eine integrierte Modellierung von Quantenalgorithmen via Drag & Drop eine hilfreiche Ergänzung.

Definition der Darstellungssprache: In [Ley19] befinden sich momentan keinerlei Skizzen oder Grafiken. Falls jedoch in Zukunft Skizzen in ein Muster integriert werden sollen, ist dies im Pattern Atlas mittels der Markdown-Funktionalität bereits möglich.

Muster verfassen: Die Erstellung einer Mustersprache erfolgt kollaborativ und iterativ [FBBL15]. Das heißt, dass während der Erstellung der Mustersprache die Autoren kontinuierlich an den Mustern arbeiten und diese verbessern und ergänzen. Außerdem werden die Muster von Dritten beurteilt, um weitere Meinungen zur Verständlichkeit, Darstellung und Korrektheit der Muster miteinbeziehen zu können. Dieser Überarbeitungs- und Korrekturprozess erfordert eine einfache und schnelle Möglichkeit der Bearbeitung der Muster und im Optimalfall eine Diskussionsplattform, um Unklarheiten auszuräumen. Im Falle von Quantencomputing-Mustern muss es also möglich sein, die inkludierten mathematischen Formeln und Quantengatterdarstellungen einfach und schnell anpassen zu können. Quantenalgorithmen sollen im Pattern Atlas direkt bearbeitet werden können, damit ein wiederholtes einlesen vermieden wird. Des Weiteren soll es möglich sein ausgewählte Teile eines Quantenschaltkreises kommentieren und diskutieren zu können.

Überarbeitung der Mustersprache: Im Pattern Atlas ist es bereits möglich jederzeit Verknüpfungen zwischen unterschiedlichen Mustern hinzuzufügen oder zu entfernen. Somit wird diese Phase des Mustererstellungsprozesses bereits vollständig unterstützt.

Zusammengefasst ergibt sich die folgende Anforderungsliste:

- Mathematische Formeln müssen einfach mittels der LaTeX-Notation definiert werden können.
- Quantenschaltkreise die mittels der gängigen Bibliotheken Qcircuit [Bry20] und Quantikz [Qua19] erstellt wurden, sollen direkt in den Pattern Atlas übernommen werden können.
- In OpenQASM implementierte Algorithmen sollen direkt in den Pattern Atlas übernommen werden können, sodass eine grafische Repräsentation des Algorithmus als Gatterdarstellung in der Musterdetailansicht zu sehen ist.

- Es muss möglich sein erstellte Quantenschaltkreise kollaborativ kommentieren und diskutieren zu können.
- Mathematische Formeln, Quantenschaltkreise und OpenQASM Implementierungen müssen einfach und schnell zu bearbeiten sein.
- Es soll möglich sein mittels Drag & Drop Quantenschaltkreise zu modellieren.

5.2 Analyse existierender Modellierungstools

Garhwal et al. [GGA19] listen eine Vielzahl unterschiedlicher Sprachen und Tools zur Implementierung und Darstellung von Quantenalgorithmen. Um einen Überblick über die momentan verfügbaren Tools und deren Umfang zu erhalten, wurde eine Toolanalyse durchgeführt. Ziel der Analyse war es, aus der Vielzahl von Tools, die für dieses Projekt relevanten Tools herauszufiltern. Es sollen also solche Tools identifiziert werden, welche bei der Umsetzung der zuvor herausgearbeiteten Anforderungen helfen. Entsprechend der Anforderungen soll ein Tool gefunden werden, mit welchem Quantenschaltkreise via Drag & Drop erstellt und gespeichert werden können. Des Weiteren soll ein Tool gefunden werden, welches in OpenQASM implementierte Algorithmen grafisch als Quantenschaltkreis darstellen kann. Somit müssen 2 Toolanalysen mit unterschiedlichem Fokus und Kriterien durchgeführt werden. Zur Suche von Tools wurden die in [Qua20b] [ZLC00] [GGA19] vorgestellten Tools einbezogen.

Analyse für Drag & Drop Quantenschaltkreissimulatoren

Mittels eines Quantenschaltkreises lassen sich Quantenberechnungen darstellen. Eine Berechnung besteht dabei aus einer Sequenz von Quantengattern. Zur Erstellung und Simulation solcher Modelle gibt es Tools, die es ermöglichen einen Quantenschaltkreis zu konstruieren. Da es eine Vielzahl dieser Tools gibt, werden diese anhand der im Folgenden definierten Kriterien bewertet. Der Pattern Atlas ist ein Open-Source-Projekt und kann als solches nur andere Open-Source-Anwendungen integrieren. Deshalb stellt die *Lizenz* der Tools ein sehr wichtiges Kriterium dar. Das zweite Kriterium ist das *Format*, welches das Tool für den Import und Export der Quantenalgorithmen benutzt. Es soll möglich sein die erstellten Algorithmen zu speichern und mit anderen Nutzern zu teilen. Das dritte Kriterium stellt der *Umfang der Drag & Drop Funktion* dar. Hier wird in die Kategorien „nicht verfügbar“ ○, „verfügbar mit kleinem Funktionsumfang“ ◑ und „verfügbar und umfangreich“ ● unterschieden. Das letzte Kriterium stellt die *Kompatibilität mit OpenQASM* dar. Im Optimalfall soll es möglich sein in OpenQASM implementierte Algorithmen in das Tool zu importieren, sodass der Algorithmus als Gatterdarstellung dargestellt wird.

In Tabelle 5.1 werden die Ergebnisse der Toolanalyse für Drag & Drop Quantenschaltkreissimulatoren dargestellt. Sowohl Quantum Inspire [QuT20] als auch QUI [Mel20] können aufgrund ihrer Lizenz nicht für das Musterrepository verwendet werden. Bei dem Quantum Circuit Inspect [unb14] handelt es sich um ein JSFiddle Skript. Es verfügt über keine Lizenzinformationen und über keine Möglichkeit gespeicherte Quantenschaltkreise zu exportieren oder importieren. Bei der Auswertung des Import- und Exportformats, fällt auf, dass es keinen Standard gibt, sondern jede Anwendung ein unterschiedliches Format verwendet. Strawberry Fields [Xan20] speichert die erstellten Quantenschaltkreise mittels Cookies ab und bietet keine Exportfunktion. Sowohl

Tool	Lizenz	Format	Drag & Drop Umfang	OpenQASM-Kompatibilität
Strawberry Fields [Xan20]	Apache 2.0	Cookies	◑	○
Quirk [Str19]	Apache 2.0	URL / JSON	●	◑
Quantum Circuit Inspector [unb14]	-	-	◑	○
Quantum Inspire [QuT20]	Kommerziell	cQASM	○	○
Quantum Circuit Simulator [wyb17]	Apache 2.0	JSON	◑	○
QUI [Mel20]	Kommerziell	Custom	●	○

Tabelle 5.1: Analyse von Tools zur Darstellung von Quantenalgorithmen.

Quirk [Str19] als auch der Quantum Circuit Simulator [wyb17] bieten die Möglichkeit erstellte Quantenschaltkreise als JSON zu exportieren und zu importieren. Quirk bietet des Weiteren das Teilen mittels URL an. Hier werden alle im Quantenschaltkreis enthaltenen Informationen in der exportierten URL als URL-Parameter gespeichert. Beim Umfang des Drag & Drop Menüs unterscheiden sich die Anwendungen sehr stark. Quantum Inspire ermöglicht es Codesnippets der einzelnen Gatter zu kopieren, welche als cQASM zu einem Quantenschaltkreis zusammengesetzt werden können. Strawberry Fields, Quantum Circuit Inspector und Quantum Circuit Simulator bieten eine einfache Drag & Drop Oberfläche, mit welcher die wichtigsten Gatter zu einem Quantenschaltkreis zusammengesetzt werden können. Quirk und QUI können sich hier von den anderen Tools absetzen. Sie zeichnen sich durch eine deutlich umfangreichere Auswahl aus: Quirk erlaubt das Erstellen eigener Gatter und kann Quantenschaltkreise mit bis zu 16 Qubits simulieren. Qui unterstützt nur Quantenschaltkreise bis zu 5 Qubits. Das letzte Kriterium wird von keinem der Tools erfüllt. Mit Hilfe von Quantum Circuit [Qua20a] ist es jedoch möglich einen OpenQASM-Algorithmus in eine Quirk-URL zu transformieren, welche den Algorithmus darstellt. Quantum Circuit ist ein Simulationstool, welches in OpenQASM implementierte Quantenschaltkreise simulieren und in vielen in der Quantencomputing Domäne verwendeten Programmiersprachen exportieren kann.

Unter Einbeziehung aller Kriterien stellt Quirk [Str19] das geeignetste Tool zur Erstellung von Quantenschaltkreisen mittels Drag & Drop dar. Leider verfügt keines der Tools über eine vollständige OpenQASM-Integration. Abbildung 5.2 zeigt die Benutzeroberfläche von Quirk. Es wird der Quantenschaltkreis der Grover Search dargestellt.

Analyse für Tools zur grafischen Darstellung von OpenQASM

Im Musterrepository soll es möglich sein in OpenQASM implementierte Algorithmen zu integrieren, welche dann grafisch dargestellt werden. Da es eine Vielzahl von Tools gibt, die eine solche Transformation ermöglichen, müssen diese mittels klar strukturierte Kriterien miteinander verglichen werden. Wie schon bei der Analyse der Tools zur Erstellung von Quantenschaltkreisen, stellt die *Lizenz* das erste wichtige Kriterium dar. Der Pattern Atlas ist ein Open-Source-Projekt und kann

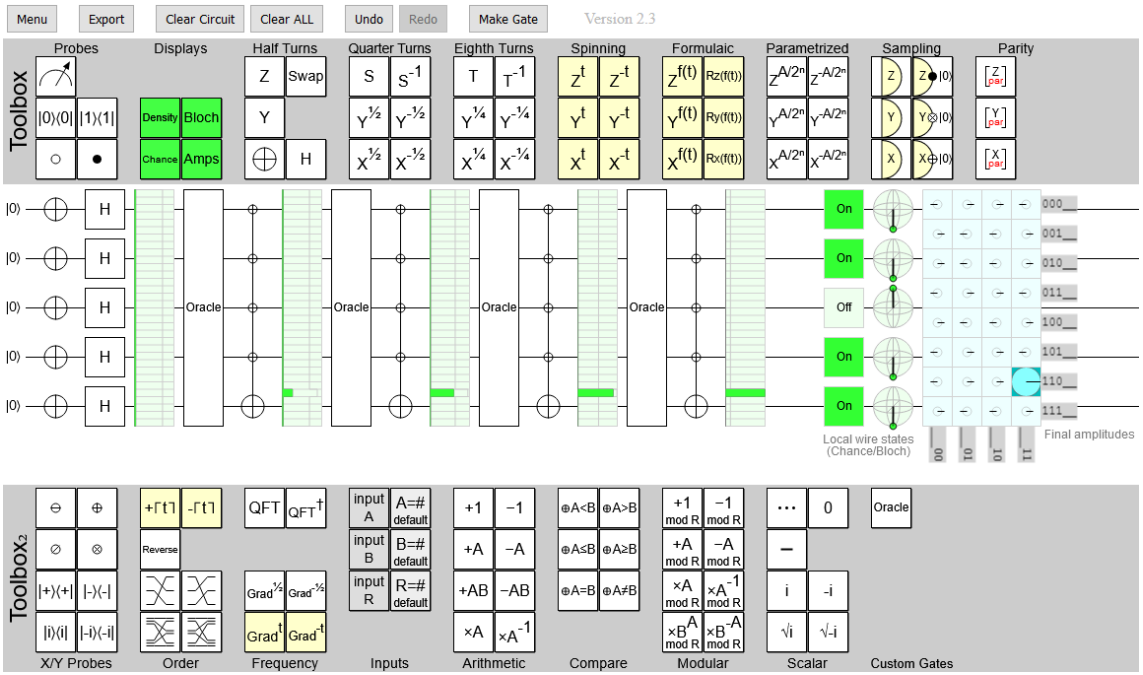


Abbildung 5.2: Quirk Benutzeroberfläche; dargestellt ist der Quantenschaltkreis der Grover Search [Str19]

als solches nur andere Open-Source-Anwendungen integrieren. Das zweite Kriterium stellt das *Eingabeformat* dar. Dies sollte OpenQASM sein. Das nächste Kriterium stellt das *Exportformat zur grafischen Darstellung* von Quantenalgorithmen dar. Im Optimalfall sollen Quantenschaltkreise als SVG exportiert werden können.

In Tabelle 5.2 werden die Ergebnisse der Toolanalyse zur grafischen Darstellung von OpenQASM-Implementierungen dargestellt. Alle Tools außer Qasm2image [Sua18] erfüllen das Open-Source-Kriterium. Qasm2image ist unter der CeCill -B Lizenz veröffentlicht, welche ein sehr starkes Copy-Left erfordert und ist somit nicht für den Pattern Atlas geeignet. Qiskit basiert auf OpenQASM, Algorithmen werden jedoch in Python definiert. Alle anderen Anwendungen unterstützen OpenQASM als Eingabeformat. Bei der Auswahl der grafischen Exportformate unterscheiden sich die Anwendungen jedoch deutlich. Während Qasm-Circuit-Preview [El 20] und Qiskit [IBM20b] nur den Export als PNG anbieten, können Qasm2image und Quantum Circuit die Quantenschaltkreise als SVG exportieren. Quantum Circuit ermöglicht noch viele weitere Exportformate die zur Erweiterbarkeit beitragen. So können Quantenschaltkreise auch als pyQuil, Quil, Qiskit, Quirk, TensorFlow Quantum, Qsharp und QuEST exportiert werden [Qua20a]. Die Exportfunktion als Quirk-URL könnte für eine weiterreichende Modellierungsfunktion nützlich sein, insbesondere falls Quantum Circuit um eine Quirk-Importoption erweitert wird. Des Weiteren ist anzumerken, dass Qiskit und Quantum Circuit die einzigen Projekte sind, die noch aktiv weiter entwickelt werden.

Quantum Circuit [Qua20a] stellt sich unter Einbeziehung aller Kriterien als am besten geeignet heraus. Quantum Circuit ist ein aktives Open-Source-Projekt, welches OpenQASM importieren und als SVG darstellen kann. Des Weiteren werden viele andere Exportformate inklusive Quirk (siehe 5.2) unterstützt.

Tool	Lizenz	Eingabeformat	Exportformat zur grafischen Darstellung
Qasm-Circuit-Preview [El 20]	MIT	OpenQASM	PNG
Qasm2image [Sua18]	CeCill -B	OpenQASM	SVG & PNG
Qiskit [IBM20b]	Apache 2.0	Python	PNG
Quantum Circuit [Qua20a]	MIT	OpenQASM	SVG

Tabelle 5.2: Analyse von Tools zur Darstellung von OpenQASM.

5.2.1 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen an das Projekt folgen den Definitionen zur Sicherstellung von Softwarequalität in ISO 25010 [Sof19]. Da das Projekt das Rendern von LaTeX-Code und Grafiken beinhaltet, muss im Bereich der Performanz auf ein akzeptables Zeitverhalten geachtet werden. Der Nutzer soll keinesfalls aufgrund von unnötigen Renderoperationen warten müssen. Des Weitern stellen die Wartbarkeit und Erweiterbarkeit wichtige Faktoren da. Da es sich um einen Prototypen in einem größeren Projekt handelt, sollen die Abhängigkeiten nicht zu eng sein, sodass es möglich ist die einzelnen Komponenten anzupassen und wieder zu verwenden. Die gute Bedienbarkeit des Pattern Atlas soll erhalten bleiben, sodass der Nutzer einfach und intuitiv auf die neuen Features zugreifen kann.

6 Konzept zur Darstellung von Quantenalgorithmen im Pattern Atlas

In diesem Kapitel werden Konzepte zur Lösung der im vorherigen Kapitel abgeleiteten Anforderungen für den Pattern Atlas vorgestellt.

6.1 Eingabe von mathematischen Formeln

Das Ziel ist es, eine einfache und intuitive Eingabe für mathematischen Formeln zur Beschreibung von Quantenzuständen in den Pattern Atlas zu integrieren. Die Eingabe soll im Markdown Editor der Weboberfläche erfolgen. Dieser ermöglicht eine benutzerfreundliche Eingabe mit vielen Optionen zur Textdarstellung und Formatierung. Da die meisten Benutzer des Pattern Atlas mit LaTeX vertraut sind, soll das Wechseln in den Mathematikmodus analog zu LaTeX funktionieren. Der Start und das Ende der Formel in derselben Zeile werden durch ein $ indiziert. Soll die Formeln alleinstehend in der nächsten Zeile stehen, wird der Start und das Ende durch $$ gekennzeichnet. Die Formeleingabe selbst soll identisch zu der Eingabe in LaTeX sein, sodass Formeln aus LaTeX auch direkt mittels Kopieren & Einfügen integriert werden können.

Zur schnellen Berechnung von LaTeX-Formeln in Webapplikationen wird zumeist MathJax [Num20] oder KaTeX [Aca20] verwendet [tex18]. Beide Projekte unterstützen gemäß ihrer Dokumentation alle in [Ley19] benötigten Zeichen. KaTeX kann Formeln um ein vielfaches schneller darstellen als MathJax, hat aber weniger Eingabe- und Ausgabeoptionen [Int20] [tex18] [Aca20]. So unterstützt MathJax beispielsweise nicht nur TeX, sondern auch MathML [Num20].

6.2 Konzept zur Darstellung von Quantenschaltkreisen

Entsprechend der in Kapitel 5 abgeleiteten Anforderungen soll es möglich sein ein in LaTeX erstellten Quantenschaltkreis einfach in den Pattern Atlas zu integrieren. Der Pattern Atlas beschränkt sich auf die Darstellung von Quantenschaltkreisen, welche mit Qcircuit [Bry20] oder Quantikz [Qua19] erstellt wurden. Sowohl Qcircuit als auch Quantikz sind LaTeX-Pakete, welche das Erstellen von Quantenschaltkreisen mittels in den Paketen definierten Befehlen ermöglichen. Der Nutzer soll die Möglichkeit haben, den in LaTeX erstellten Code zu kopieren und im Pattern Atlas einzufügen. Da in Mathjax und KaTeX keine Pakete importiert werden können, erfordert die Darstellung von Quantenschaltkreisen mit LaTeX eine vollständige LaTeX-Distribution. Demzufolge ist die Umsetzung um einiges umfangreicher und komplizierter. Die folgenden Probleme müssen gelöst werden:

1. Wie werden die zu rendernden Quantenschaltkreise identifiziert?

2. Wie, wo und in welchem Format werden die Quantenschaltkreise gerendert?
3. Wie werden die gerenderten Quantenschaltkreise gespeichert?
4. Wie erfolgt die Darstellung der Quantenschaltkreise?
5. Wie können Quantenschaltkreise einfach bearbeitet werden?

6.2.1 Systemarchitektur

Um diese Probleme zu lösen, muss der Pattern Atlas um einige Komponenten und Services erweitert werden. In Abbildung 6.1 wird die konzeptionelle Systemarchitektur zur Umsetzung der Quantencomputing-Features vorgestellt. Der Pattern Atlas besteht aus einem in Angular [Goo20] implementierten Frontend und einem in Java mit Spring-Boot [VMw20] implementierten Backend. Somit ergibt sich mit der PostgreSQL-Datenbank [Gro20] im Backend eine klassische Webanwendung nach dem Model-View-Controller-Prinzip. Der Nutzer greift über einen Web-Browser auf das Frontend des Pattern Atlas zu. Sobald ein Nutzer ein Muster speichert oder einen Kommentar verfasst, werden die entsprechenden Backend-Controller aufgerufen. Diese führen nun die notwendigen Services aus, um die erhaltenen Anfragen zu verarbeiten. Kommentare und Änderungen an den Mustern werden persistent in der Datenbank gespeichert und Muster werden auf Quantenschaltkreise überprüft. Wenn ein Quantenschaltkreise gefunden wird, wird dieser an den LaTeX-Rendering-Service gesendet, welcher ein Bild zurück gibt, welches dann vom Image-Service gespeichert wird. Um eine gute Modularität zu erhalten, wird der LaTeX-Rendering-Service nicht in das Backend integriert, sondern als alleinstehender Service umgesetzt. So können in Zukunft auch andere Anwendungen auf den Service zugreifen. Die Aufgaben der einzelnen Controller und Services und deren Interaktionen werden in den folgenden Abschnitten genauer erläutert.

6.2.2 Identifikation der Quantenschaltkreise

Quantenschaltkreise werden vom Pattern-Render-Service identifiziert. Da Quantenschaltkreise fließend in die Musterbeschreibungen integriert werden sollen, muss es möglich sein diese inmitten von Text zu erkennen. Hierzu ist es notwendig einen Blick auf exemplarische Quantikz- und Qcircuit-Beispiele zu werfen. Wie in Listing 6.1 dargestellt, wird eine Quantikz-Umgebung mit „\beginquantikz“ initiiert und mit „\endquantikz“ beendet. Dies ist eindeutig identifizierbar. Die Erkennung von Qcircuit hingegen ist nicht so einfach. Zwar ist es, wie in Listing 6.2 dargestellt wird, einfach den Startpunkt „\Qcircuit“ zu erkennen, jedoch wird das Ende der Gatterdarstellung durch eine „} “ gekennzeichnet. Da die geschweifte Klammer ein recht häufig verwendetes Symbol ist und die Beschreibung des Quantenschaltkreises selbst schon mehrere geschweifte Klammern enthält, kann das Ende nicht ausschließlich durch diese identifiziert werden. Um ein eindeutiges Ende zu kennzeichnen, müssen Qcircuit-Darstellungen im Pattern Atlas mit „end}“ anstatt von „} “ beendet werden.

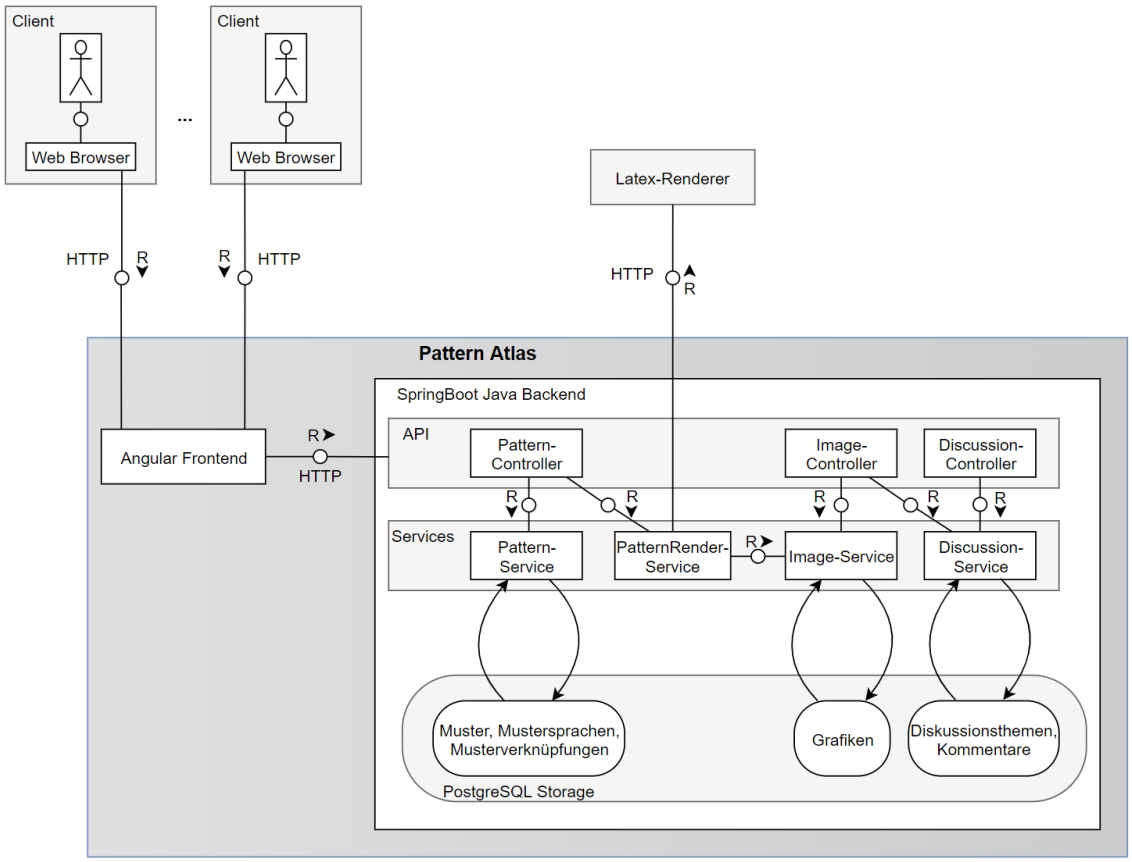


Abbildung 6.1: Konzeptionelles FMC-Architekturdiagramm [AR05] zur Integration der Quantencomputing-Features in den Pattern Atlas

Listing 6.1 Quantikz-Beispiel [Qua19]

```
\begin{quantikz}
        \lstick{$\ket{0}$} & \gate{H} & \ctrl{1} & \gate{U} & \ctrl{1} & \swap{2} & \ctrl{1} & \qw \\
        \lstick{$\ket{0}$} & \gate{H} & \targ{} & \octrl{-1} & \control{} & \qw & \octrl{1} & \qw \\
        &&&&&\targX{} & \gate{U} & \qw
\end{quantikz}
```

Listing 6.2 Qcircuit-Beispiel mit end } Tag [Bry20]

```
\Qcircuit @C=1em @R=.7em {
        & \ctrl{2} & \targ & \gate{U} & \qw \\
        & \qw & \ctrl{-1} & \qw & \qw \\
        & \targ & \ctrl{-1} & \ctrl{-2} & \qw \\
        & \qw & \ctrl{-1} & \qw & \qw end}
```

6.2.3 Renderprozess der Quantenschaltkreise

Wie in Abbildung 6.2 dargestellt wird, soll die Identifikation der Quantenschaltkreise im Pattern Atlas Backend stattfinden. Nachdem das Backend den Code zur Darstellung eines Quantenschaltkreises identifiziert hat, wird dieser inklusive der zum Rendern notwendigen LaTeX-Pakete an den LaTeX-Rendering-Serivce gesendet. Es ist nicht ausreichend nur den Code zur Gatterdarstellung zu senden, da sowohl Quantikz als auch Qcircuit mehrere Bibliotheken verwenden, welche explizit angegeben werden müssen. Der LaTeX-Rendering-Service erstellt nun ein neues LaTeX-Dokument, welches die angegeben Pakete und den Code zur Gatterdarstellung enthält. Das Dokument wird als PDF gerendert und anschließend als Grafik exportiert. Diese Grafik wird an das Backend zurückgegeben, welches die Grafik in das Muster integriert und dieses anschließend in der Datenbank speichert. Zuletzt wird das aktualisierte Muster an das Frontend zurückgegeben und dem Nutzer angezeigt.

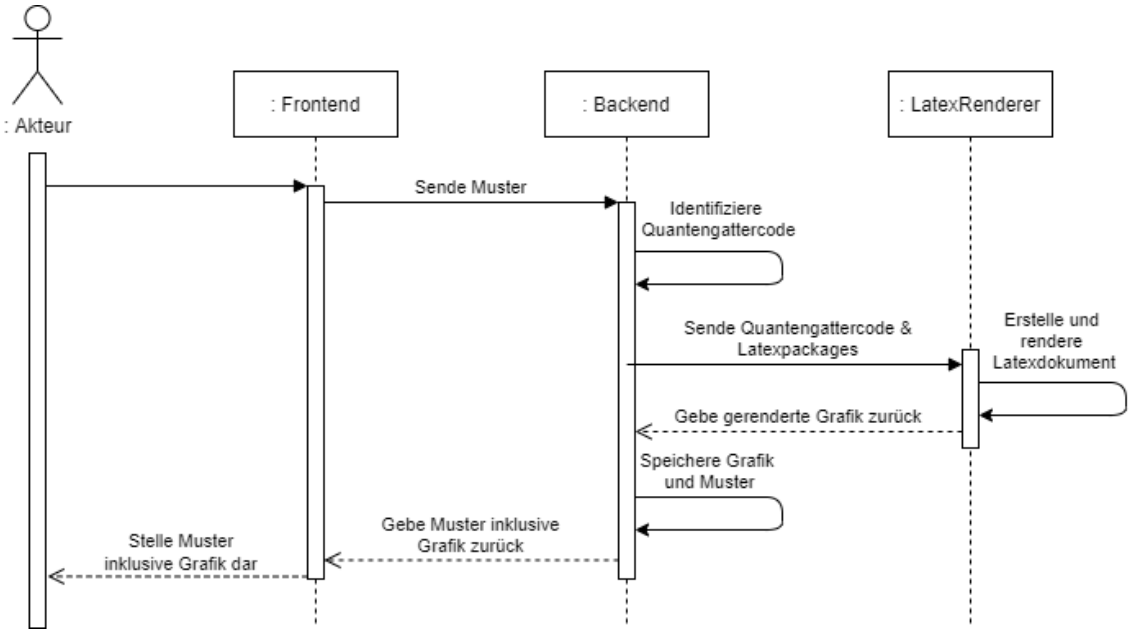


Abbildung 6.2: Sequenzdiagramm für das Rendern eines Quantenschaltkreises

6.2.4 Speicherung der Quantenschaltkreise

Die vom LaTeX-Rendering-Service empfangenen Grafiken werden in der Datenbank des Backends gespeichert.

6.2.5 Darstellung der Quantenschaltkreise

Die Quantenschaltkreise werden im Frontend als SVG dargestellt. SVGs können entweder einfach mittels des <img> Tags oder als Inline-SVG in HTML eingebunden werden. Bei der Inline-Darstellung von SVGs, wird der gesamte Inhalt des SVGs in das HTML-Dokument eingefügt.

Dadurch ergibt sich der Vorteil, dass alle Elemente des SVGs in Echtzeit ausgelesen, selektiert und bearbeitet werden können [Moz20a]. Zur Auswahl und Bearbeitung von SVG-Elementen soll d3.js [Bos20] verwendet werden.

6.2.6 Bearbeitung von Quantenschaltkreisen

Wenn ein Nutzer den LaTeX-Code für einen Quantenschaltkreis in den Pattern Atlas einfügt und anschließend das Muster speichert, soll der enthaltene LaTeX-Code gerendert werden. Anschließend soll dem Nutzer die gerenderte Grafik dargestellt werden. Angenommen der Nutzer stellt fest, dass der LaTeX-Code einen Fehler hatte soll es möglich sein, beim Bearbeiten wieder auf den ursprünglichen LaTeX-Code zuzugreifen. Demzufolge darf der LaTeX-Code nicht nach dem Rendern ersetzt werden. Der Pattern Atlas stellt momentan im Markdown-Editor beim Bearbeiten und in der Musterdetailansicht den gleichen Inhalt dar. Da im Markdown-Editor der ursprüngliche Quelltext und in der Musterdetailansicht die gerenderte Grafik angezeigt werden soll, muss sowohl das Model zum Speichern der Muster, als auch die Anzeige- und Bearbeitungslogik angepasst werden. Es muss also zwischen dem ursprünglichem Inhalt vor dem Rendern und dem gerenderten Inhalt unterschieden werden.

6.3 Konzept zur Modellierung von QASM

Zur Darstellung von OpenQASM-Code soll entsprechend der Analyse in Kapitel 5 Quantum Circuit [Qua20a] verwendet werden. Da OpenQASM-Dateien, ähnlich wie Qcircuit-Code, kein im Fließtext klar identifizierbares Ende haben, muss das Ende eines OpenQASM-Abschnitts, wie in Listing 6.3 aufgezeigt wird, mittels eines „end“ Tags identifiziert werden.

Listing 6.3 OpenQASM-Beispiel mit notwendigem „end“ Tag

```
1 OPENQASM 2.0;
2 include "qelib1.inc";
3 qreg q[2];
4 h q[0];
5 cx q[0], q[1]; end
```

Quantum Circuit importiert den identifizierten OpenQASM-Code und exportiert ihn als SVG, welches direkt im Frontend dargestellt wird.

6.4 Kommentieren und Diskutieren von Wissensartefakten

Kommentieren und Diskutieren stellt einen essentiellen Teil der kollaborativen und iterativen Entwicklung der Quantenmuster dar. Da im Pattern Atlas nicht nur Quantenschaltkreise, sondern auch andere Wissensartefakte kommentiert und diskutiert werden soll gibt einheitliche Vorgaben. Wie in Abbildung 6.3 dargestellt wird soll es möglich sein Diskussionsthemen zu einem Wissensartefakt zu erstellen. Diese sollen einen Titel, eine Beschreibung, ein Datum und einen Status haben und durch eine UUID identifiziert werden. Zu jedem Thema können Kommentare verfasst werden,

welche den vom Nutzer verfassten Text und das aktuelle Datum beinhalten. Sie werden wie auch das Diskussionsthema durch eine UUID identifiziert. Des Weiteren enthalten Kommentare die Id, des vorhergehenden Kommentars.

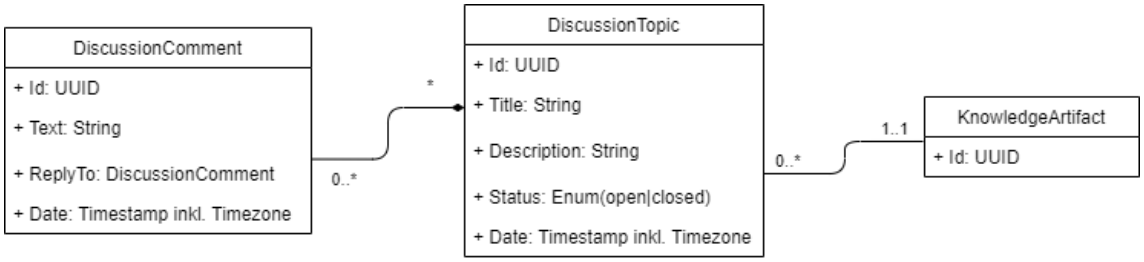


Abbildung 6.3: Auszug aus dem Klassendiagramm QC Atlas [Stu20]

6.5 Bearbeitung von Artefakten

Ziel ist es, dass im Pattern Atlas integrierte Artefakte, wie beispielsweise eine in LaTeX beschriebener Quantenschaltkreis oder ein in OpenQASM implementierter Algorithmus ohne zusätzlichen Aufwand jederzeit bearbeitet werden können. Wenn man jedoch die in Abbildung 6.4 dargestellte Datenstruktur eines Musters betrachtet, fällt auf, dass es nur ein Attribut zum Speichern aller Inhalte des Musters gibt. Da es möglich sein soll die integrierten Artefakte einfach zu bearbeiten, kann nach dem Rendern der Quelltext nicht mit der gerenderten Grafik ersetzt werden. Stattdessen muss zwischen dem Inhalt vor dem Rendern, welcher vom Nutzer bearbeitet werden kann und dem Inhalt nach dem Rendern unterschieden werden. Deshalb wird das rot hervorgehobene Attribut „RenderedContent“ hinzugefügt, in welchem der vom Backend gerenderte Inhalt gespeichert wird. Demzufolge wird auch „RenderedContent“ zur Darstellung in der Musterdetailansicht benutzt. Lediglich beim Bearbeiten eines Musters wird das ursprüngliche „Content“ Attribut dargestellt.

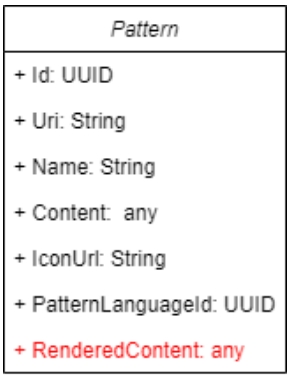


Abbildung 6.4: Erweitertes Mustermodell

6.6 Drag & Drop Integration

Die Erstellung von Quantenschaltkreisen mittels Drag & Drop ist eine einfache und schnelle Möglichkeit, um Algorithmen grafisch zu erstellen. Wie in Abschnitt 5.2 erarbeitet wurde, ist Quirk das am besten geeignete Drag & Drop Tool. Da Quirk eine eigenständige Webanwendung ist, soll Quirk nicht direkt in den Pattern Atlas integriert werden. In Abbildung 6.5 wird dargestellt, wie Quirk in Kombination mit Quantum Circuit verwendet werden kann. Quirk unterstützt zwar den Import und Export von erstellten Quantenschaltkreisen und kann somit zum kollaborativen Arbeiten an Quantenschaltkreisen grundsätzlich verwendet werden, bietet aber keine APIs dafür an. Dies führt dazu, dass Nutzer zwar als Links exportiere Quirk-Schaltkreise in den Pattern Atlas integrieren können, diese aber bei einer Änderung manuell ersetzt werden müssen. Selbiges gilt für die Verwendung in Kombination mit Quantum Circuit. Quantum Circuit, bietet zwar einen Quirk-Link des in OpenQASM implementieren Algorithmus an und aktualisiert diesen sobald der OpenQASM-Code geändert wird. Eine Änderung in Quirk, kann jedoch nicht zu einer Anpassung des OpenQASM-Codes führen. Um Quirk wie gewünscht in den Pattern Atlas zu integrieren, müsste Quirk und dessen API angepasst werden, sodass ein fließender Import-Export Übergang mit dem im Fokus stehenden Muster möglich ist. Um eine vollständige OpenQASM-Funktionalität zu erreichen, müsste des außerdem eine Funktion zur Konvertierung von Quirk zu OpenQASM erstellt werden. In diesem Projekt wird nur die einfache Integration von Quirk umgesetzt, welche es erlaubt OpenQASM-Algorithmen in Quirk zu öffnen.

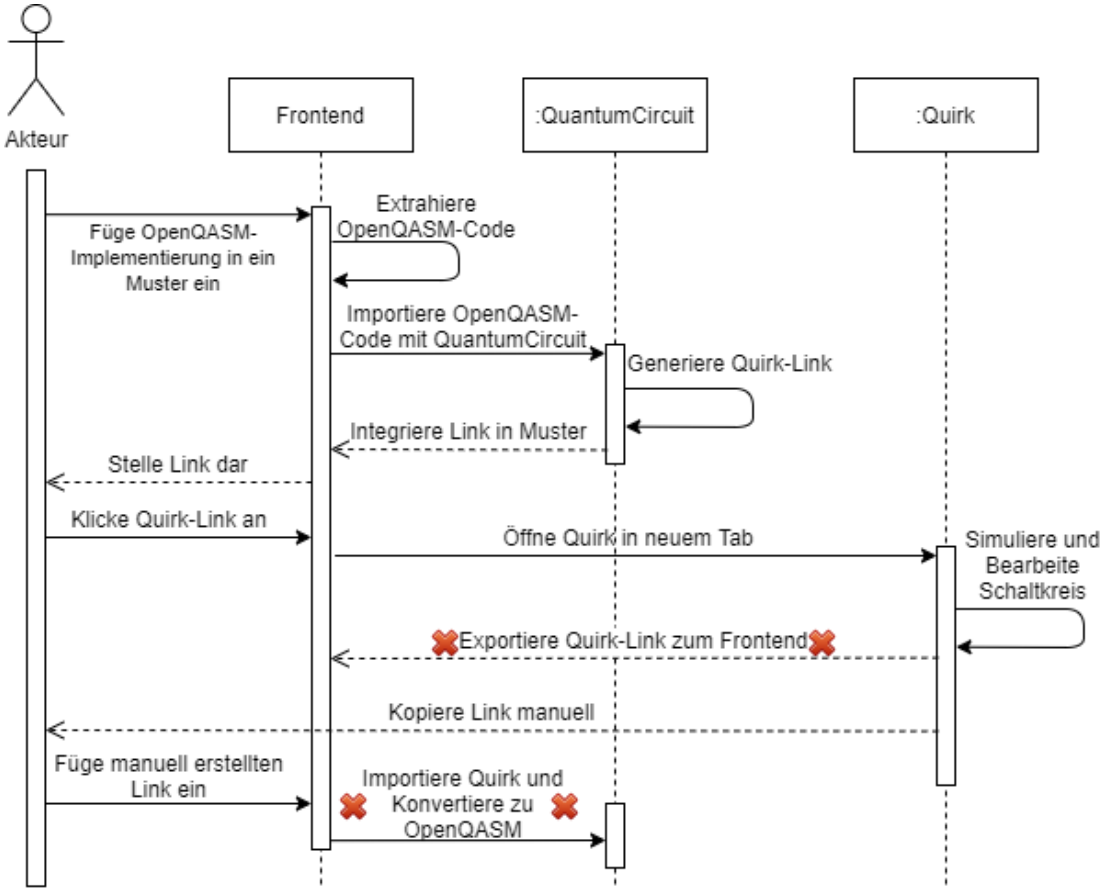


Abbildung 6.5: Sequenzdiagramm zur Darstellung des Zusammenspiels von Quantum Circuit und Quirk

7 Implementierung im Pattern Atlas

In diesem Kapitel wird die Implementierung des Prototyps vorgestellt, welcher die im vorherigen Abschnitt erarbeiteten Konzepte umsetzt. Zuerst wird die Implementierung des Pattern Atlas vorgestellt und anschließend wird auf die Umsetzung des LaTeX-Rendering-Services eingegangen. In Abbildung 7.1 wird das Komponentendiagramm des Pattern Atlas dargestellt. Es zeigt die in den folgenden Abschnitten genauer beschriebenen Abhängigkeiten und Schnittstellen zwischen den einzelnen Komponenten auf und dient als Übersicht für das gesamte Kapitel.

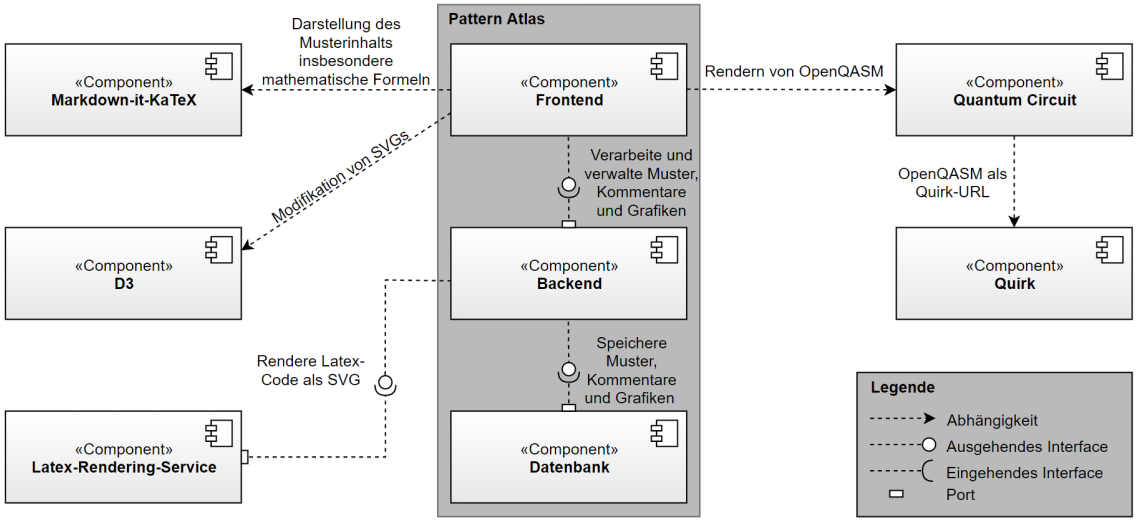


Abbildung 7.1: Komponentendiagramm des Pattern Atlas

7.1 Umsetzung der Darstellung von Quantenformeln

Zu Beginn dieser Arbeit wurden Formeln im Pattern Atlas mit Markdown-it-KaTeX [way16] dargestellt. Markdown-it-KaTeX ermöglicht es mathematische Formeln in Markdown zu integrieren. Zur Darstellung wird KaTeX [Aca20] verwendet, welches auf TeX und LaTeX basiert. Markdown-it-KaTeX parst mathematische Formeln gemäß den von pandoc [pan20] gesetzten Richtlinien. Demnach werden alle Zeichen zwischen zwei $-Zeichen als TeX-Math behandelt. Das eröffnende $-Zeichen muss ein direkt folgendes Leerzeichen haben, während das schließende $-Zeichen ein direkt vorhergehendes Leerzeichen haben muss. Das Problem mit der im Pattern Atlas integrierten Version ist, dass einige für Quantenalgorithmen nötig Zeichen, wie „$\underbrace{\quad}$“ nicht unterstützt werden.

Da KaTeX entsprechend der Dokumentation und des durchgeführten Vergleichs mit Mathjax für das Projekt gut geeignet ist und bereits im Projekt integriert ist, soll weiterhin daran festgehalten werden und das existierende Problem gelöst werden. Laut der KaTeX-Dokumentation werden alle für die

Listing 7.1 JSON-Darstellung des Musterinhaltes des Uniform-Superposition-Musters [Ley19]

```
{"content":
        {"Icon":"![enter image description here](http://www...)",
        "Next":"Creating uniform superposition makes use of initialization ...",
        "Intend":"Typically, the individual qbits of a quantum register ...",
        "Context":"One origin of the power of quantum algorithms ...",
        "Solution":"Uniform superposition is achieved by initializing ...",
        "Known uses":"Most algorithms make use of uniform superposition",
        "Driving Question":"How can an equally weighted ..."}
}
```

Darstellung von mathematischen Formeln zur Beschreibung von Quantenzuständen benötigten Zeichen unterstützt. Somit lässt sich Schlussfolgern, dass die mangelhafte Zeichenunterstützung von Markdown-it-KaTeX nicht direkt an KaTeX, sondern an der Umsetzung von Markdown-it-KateX selbst liegt. Um einen Installationsfehler auszuschließen, war der nächste Schritt zu überprüfen, ob Markdown-it-KateX entsprechend der Installationsanleitung integriert wurde. Dabei hat sich herausgestellt, das Markdown-it-Katex zwar richtig installiert wurde, die verwendeten KaTeX-Skripte allerdings überholt sind. Eine Anpassung der Skripte auf die aktuelle KaTeX-Version hat das Problem nicht gelöst. Die Integration der von MicroDroid geforkte Version von Markdown-it-KaTeX [Mic19], welche KaTeX 0.11.1 anstatt von Version 0.6.0 verwendet führte schließlich zum Erfolg. Alle Zeichen können problemlos gerendert werden.

7.2 Umsetzung der Darstellung von Quantenschaltkreisen

7.2.1 Darstellung von LaTeX-Code als SVG

Der Pattern Atlas setzt die natürlichen CRUD-Verben: CREATE, READ, UPDATE & DELETE als Schnittstellen für die Verwendung von Muster um. Demzufolge gibt es zwei Möglichkeiten eine Speicheroperation auf einem Muster durchzuführen. Eine Speicheroperation kann auf die initiale Erstellung oder die Bearbeitung eines Musters folgen. Bei jedem Abspeichern muss überprüft werden ob, das zu speichernde Muster Code enthält, welcher gerendert werden muss. Sowohl bei einer initialen Erstellung, als auch bei einer Bearbeitung eines Musters enthält der POST- bzw PUT-Request ein „content“-Feld. In diesem werden alle Attribute der Mustersprache und deren Werte gespeichert. In Listing 7.1 wird als Beispiel eine gekürzte Version des „content“-Feldes des Uniform-Superpositon-Musters [Ley19] dargestellt. Der Pattern-Controller, welcher die Schnittstelle für das Abfragen, Speichern und Löschen von Mustern bildet, leitet das empfangene Muster an den Pattern-Render-Service weiter. Dieser wandelt alle Inhalte des „content“-Feldes in einen String um, sodass Suchoperationen einfach auf diesem durchgeführt werden können.

Als nächstes wird der Content-String auf alle Vorkommnisse von Quantikz- oder Qcircuit-Code überprüft. Im Falle eines Fundes, wird der LaTeX-Code aus dem Content-String extrahiert und auf das Rendern mittels des LaTeX-Rendering-Services vorbereitet. Da die Muster via JSON an das Backend verschickt werden, muss auf die dadurch entstehenden Limitierungen geachtet werden. Neue Zeilen werden als \n, Tabs als \t und Backslashes als \\ dargestellt. In LaTeX führen diese

Codierungen zu Ausführungsfehlern und müssen demzufolge angepasst werden. Da neue Zeilen und Tabs zum Rendern der Quantenschaltkreisen nicht relevant sind, werden diese durch ein Leerzeichen ersetzt. Backslashes hingegen sind essentiell für Qcircuit- und Quantikz-Befehle. Um wieder auf den vom Nutzer eingegebenen Code zurückzugelangen, werden alle doppelten Backslashes durch einfache Backslashe ersetzt. Der resultierende String wird mitsamt der entsprechenden Quantikz- oder Qcircuit-Paketdefintionen und dem gewünschten Rückgabeformat (SVG) an den LaTeX-Rendering-Service (siehe 7.4) geschickt. Sobald das gerenderte SVG als Byte-Array empfangen wurde, wird dieses in die Datenbank eingepflegt. Das Backend speichert die empfangenen Bilder in dem in Abbildung 7.2 dargestellt Format in der PostgreSQL-Datenbank. Nun muss das Bild

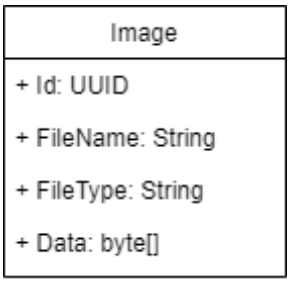


Abbildung 7.2: Klassendiagramm zum Speichern von Bildern

anstatt des LaTeX-Codes in den Content-String eingefügt werden. Da aufgrund der Anpassbarkeit, Übersichtlichkeit und Größe es suboptimal wäre den Quellcode des SVGs direkt in das Muster zu integrieren, wird stattdessen die Id des Bildes angegeben, über welche dann auf das Bild zugegriffen werden kann. Anschließend wird der überprüfte und wenn nötig bearbeitete Content-String an den Pattern-Controller zurückgegeben und im Attribut „renderedContent“ des Musters gespeichert. Somit kann klar zwischen dem ursprünglichem, vom Nutzer verfassten Text, und dem vom Backend überprüften und bearbeiteten Text unterschieden werden.

7.2.2 Erkennung bereits gerenderter Quantenschaltkreise

In der zuvor beschriebenen Erkennungsmethode zum Rendern von Quantenschaltkreisen werden alle im Content-Feld enthaltenen Quantikz- oder Qcircuit-Vorkommnisse identifiziert und gerendert. Wenn ein Nutzer ein existierendes Muster bearbeitet, jedoch keine Änderungen an den im Muster enthaltenen LaTeX-Darstellungen vornimmt, identifiziert und rendert das Backend diese trotzdem. Dies ist sehr ineffizient und führt aufgrund der deutlich merkbaren Renderzeit des LaTeX-Rendering-Services zu unnötiger Wartezeit und schlechter Nutzbarkeit. Um dieses Problem zu lösen, wird das empfangene Muster mit dem in der Datenbank gespeicherten Muster abgeglichen. Falls Codevorkommnisse identisch und somit unverändert sind, muss der LaTeX-Code nicht erneut gerendert werden.

7.2.3 Darstellung der Quantenschaltkreisen im Frontend

Wie bereits in Abschnitt 7.2.1 beschrieben wird, wird der Quellcode des SVGs inline eingebunden. Da es möglich sein soll mehrere SVGs in ein Muster zu integrieren müssen einige Anpassungen vorgenommen werden, damit diese fehlerfrei dargestellt werden können. Bei der Erstellung der

SVGs werden bestimmte Symbole explizit definiert und über eine Id identifiziert. Das Problem liegt nun darin, dass das zum Rendern verwendete Tool die Ids standardmäßig für jedes Bild einzeln von 1 an hoch zählt. Demzufolge existiert mehrfach die gleiche Id sobald mehrere SVGs auf einer Seite dargestellt werden. Zur Lösung dieses Problems wird jedem SVG die ImageId aus dem Backend zugewiesen. Anschließend werden alle Symbol-Ids und deren Verwendungspositionen nach dem Schema *Name* + *ImageId* + *Nummer* umbenannt. So wird beispielsweise aus „glyph-1“ → „glyph*ID*-1“. Somit hat jedes SVG individuelle Ids und es ist problemlos möglich unbegrenzt viele Quantenschaltkreise in einem Muster darzustellen.

7.2.4 Kommentieren der Quantenschaltkreise

Das Kommentieren und Diskutieren von Wissensartefakten ist ein zentraler Punkt des iterativen Mustererstellungsprozesses. Musterautoren und Nutzern soll es möglich sein bestimmte Bereiche eines Quantenschaltkreises zu markieren und diese zu kommentieren. Anschließend soll auf den Kommentar geantwortet werden können. Die in 6.3 dargestellten allgemeinen Attribute zum Kommentieren von Wissensartefakten sind allerdings nicht ausreichend, um die SVGs der Quantenschaltkreise zu kommentieren. Es soll nicht nur möglich sein die Grafik zu kommentieren, sondern explizit einen Bereich zu markieren, welcher gekennzeichnet und kommentiert werden soll. Außerdem muss der Kommentar mit dem zugehörigen Bild verknüpft werden. Somit wird das Klassendiagramm, wie in Abbildung 7.3 dargestellt, erweitert. Es werden die x- und y-Koordinaten des Startpunktes des Kommentarbereichs gespeichert. Die Informationen zur Darstellung des Bereichs werden durch die Breite, Höhe und Farbe vervollständigt. Zuletzt wird noch die ID des zum Thema gehörigen Bildes gespeichert. Um die Kommentare klar einem Thema zuzuordnen, werden diese um die Id des zugehörigen Themas erweitert. Da in SVGs nicht nur Vektoren zur

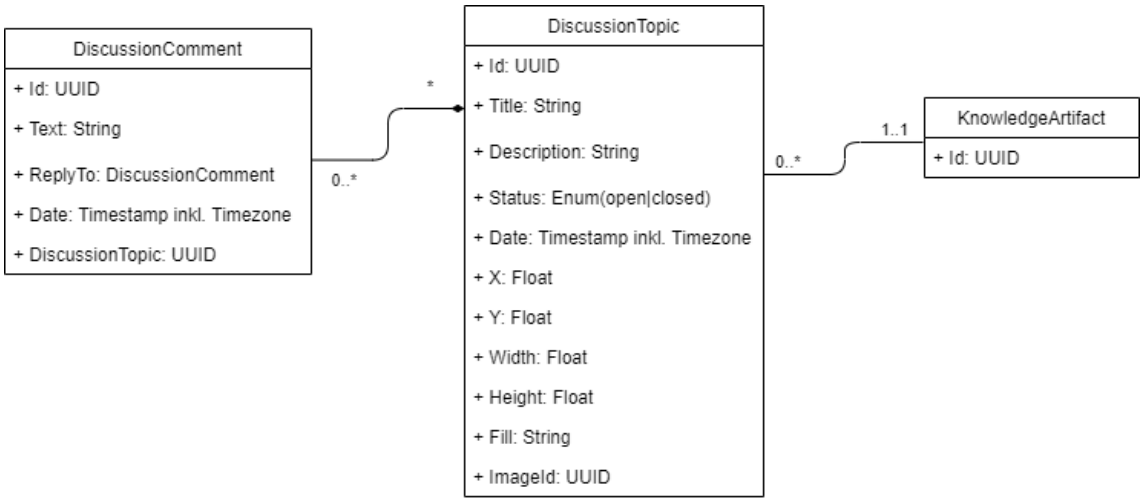


Abbildung 7.3: Angepasstes Klassendiagramm für Diskussionsartefakte

Darstellung von Objekten, sondern auch Daten gespeichert werden können, ist es möglich die Kommentare direkt in die SVGs zu integrieren.

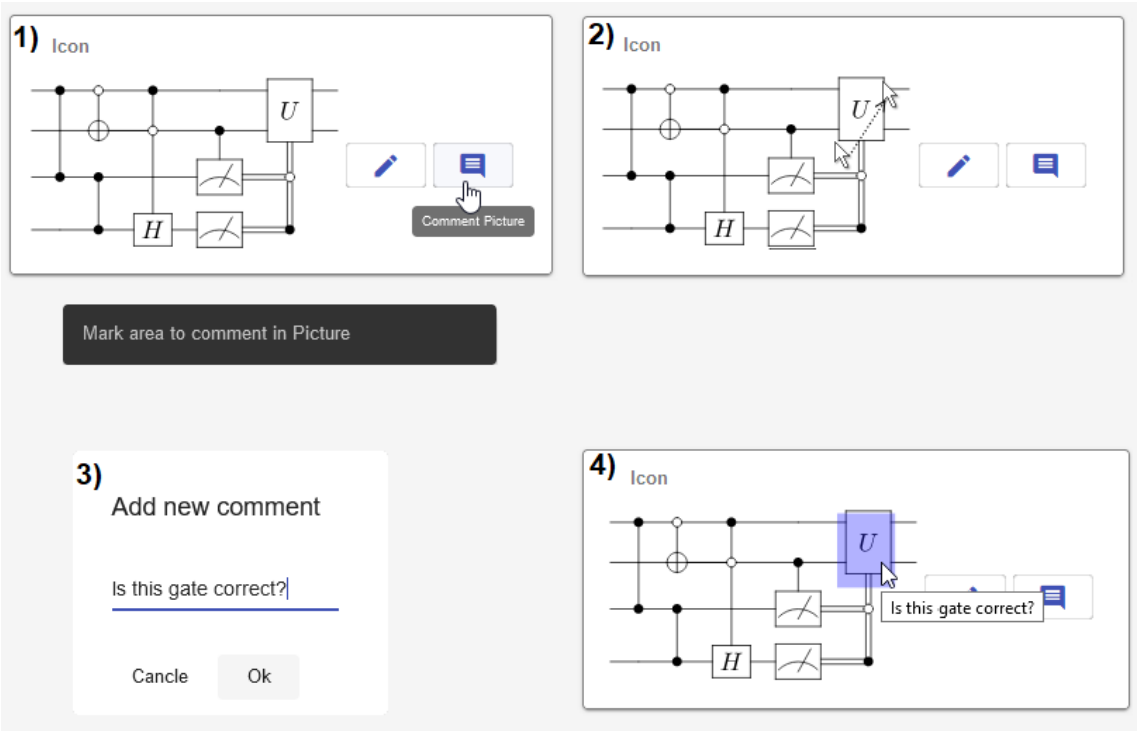


Abbildung 7.4: Prozess zum Erstellen eines Kommentars

Erstellen des Kommentarbereiches

Es soll möglich sein, nicht nur ein SVG als Ganzes zu kommentieren, sondern einen Bereich explizit auszuwählen, welcher markiert und kommentiert werden kann. In Abbildung 7.4 wird der Prozess zu Erstellung eines solchen Kommentarbereiches dargestellt. Da nicht jeder Klick im SVG mit dem Ziel getätigt wird einen neuen Kommentar zu erstellen, soll die Kommentarfunktion mit einem Button aktiviert werden müssen. Dies funktioniert wie in Schritt 1 der Abbildung 7.4 dargestellt wird. Sobald die Kommentarfunktion aktiviert wurde, bekommt der Nutzer eine Benachrichtigung, dass er nun einen Bereich markieren kann. Die Markierung soll einfach und intuitiv erstellt werden können. Eine Markieroperation soll, wie in Schritt 2 dargestellt wird, mittels eines Mousedown-Events am Startpunkt und eines Mouseup-Events am Endpunkt durchgeführt werden. Die Punkte werden anschließend zu einem Rechteck verbunden. Sobald ein Bereich markiert wurde, wird dem Nutzer ein Popup angezeigt, in welchem er seinen Kommentar schreiben kann. Nach der Fertigstellung des Kommentars ist der ausgewählte Bereich farbig hinterlegt und der Nutzer kann bei einer Mouseover-Operation den Inhalt des Kommentars als Tooltip sehen.

Integration von Kommentaren in das SVG Das Markieren von Punkten in SVGs funktioniert allerdings nicht so einfach wie das Markieren in herkömmlichen Pixeldarstellungen. SVGs haben ein eigenes Koordinatensystem, welches nicht den Bildschirmkoordinaten entspricht. So muss bei jeder Mausoperation, die Mausposition in die SVG-Koordinatenposition umgerechnet werden. Dazu muss die Document-Object-Model (DOM)-Funktion *SVGGraphicsElement.getScreenCTM()* auf dem SVG aufgerufen werden [Mic16]. Diese gibt eine Transformationsmatrix zurück, welche verwendet werden kann, um die Koordinaten eines SVG-Punkts in Bildschirmkoordinaten umzuwandeln

[Mic16]. Da das Ziel jedoch die Transformation von Bildschirmkoordinaten zu SVG-Koordinaten ist, muss diese Transformationsmatrix invertiert werden. Mit der inversen Transformationsmatrix ist es also nun möglich die vom Nutzer in Abbildung 7.4 Schritt 2 eingegebenen Mauspositionen in SVG-Koordinaten umzuwandeln. Anschließend wird der kleinere x- und y-Wert der beiden Koordinaten bestimmt und die Länge und Breite des Rechtecks berechnet. Mit diesen Informationen kann nun ein Rechteck an der vom Nutzer markierten Stelle in das SVG eingefügt werden. Das Rechteck bildet zusammen mit dem in Schritt 3 eingefügten Kommentar, das in der Datenstruktur 7.3 definierte Diskussionsthema. Das Diskussionsthema wird im Backend gespeichert.

Hinzufügen von Antworten auf einen Kommentar Bisher ist es nur möglich einen neuen Bereich zu markieren und einen neuen Kommentar für diesen zu erstellen. Dies erfüllt jedoch nicht die gesetzten Ziele des Austauschs und der Diskussion. Nutzer sollen auf bereits verfasste Kommentare reagieren und antworten können. Um dies möglichst intuitiv und übersichtlich umzusetzen, wurde entschieden, wie in Abbildung 7.5 dargestellt, ein Diskussionsfenster einzufügen, welches beim Klicken auf einen Kommentar geöffnet werden kann. Der Dialog zeigt alle zuvor verfassten Antworten in chronologischer Reihenfolge an und bietet die Möglichkeit eine weitere Antwort hinzuzufügen oder das gesamte Diskussionsthema und die zugehörige grafische Markierung zu löschen. Die Antworten entsprechen den Diskussionskommentaren aus Abbildung 7.3. Der Kommentar wird im Backend gespeichert und dem SVG hinzugefügt.

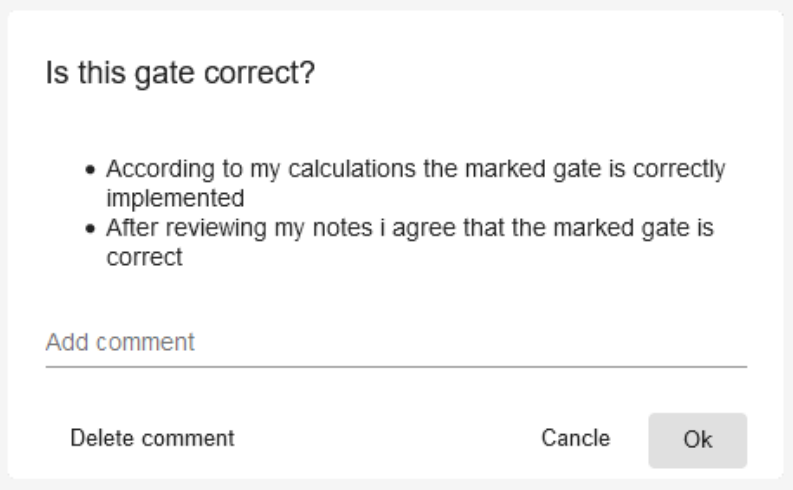


Abbildung 7.5: Diskussionsdialog

7.2.5 Erkennung bearbeiteter Diagramme

Da es in vielen Fällen vorteilhaft wäre die Kommentare nach einer kleinen Anpassung eines Quantenschaltkreises zu erhalten, soll dies ebenfalls umgesetzt werden. Andernfalls wird sobald auch nur eine kleine Änderung im LaTeX-Code durchgeführt wird, das gesamte Bild neu gerendert und alle der Grafik zugehörigen Kommentare gehen verloren. Zu erkennen, ob es sich bei dem neu zu rendernden LaTeX-Code, um ein angepasstes Diagramm, welches die Kommentare übernehmen soll, oder ein neues Diagramm an der gleichen Stelle handelt ist nicht trivial. Die Erkennung

kann entweder manuell über eine Benutzerabfrage oder automatisch, mittels beispielsweise einer Überprüfung der Ähnlichkeit des LaTeX-Codes, durchgeführt werden. In diesem Projekt wurde sich für eine automatische Überprüfung mit Hilfe der Jaccard Similarity entschieden. Mit Hilfe der Jaccard Similarity lässt sich messen, wie viele gemeinsame Wörter zwei Texte beinhalten. Die Jaccard Similarity der Wortmengen A und B lässt sich über $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$ definieren [Kos19]. Der resultierende Wert liegt immer zwischen 0 und 1. Bei einem Wert von 0, gibt es keine gemeinsamen Wörter, bei einem Wert von 1 sind die Wortmengen identisch [Ma18]. Zur Erkennung von abgeänderten Diagrammen wird 0,8 als Treshold verwendet. Der Vorteil der Jaccard Similarity ist, dass wenn beispielsweise zwei Gatter im Code vertauscht werden, die Quantenschaltung immer noch als identisch beurteilt wird, da sich die Menge der Wörter nicht ändert.

Um zu erkennen, welche Grafik angepasst wurde und welche Kommentare demzufolge von dem alten auf das neue Bild übertragen werden müssen, wird die Jaccard Similarity der in Frage stehenden LaTeX-Codeabschnitte berechnet. Dazu wird das neue empfangene Muster mit der in der Datenbank gespeicherten Version abgeglichen. Falls die Ähnlichkeit den Schwellwert von 0,8 übersteigt, werden dem neuen Bild die Diskussionsthemen und Kommentare des alten Bildes hinzugefügt. So kann der Nutzer im Anschluss das neu gerenderte Bild mitsamt der Kommentare sehen.

7.3 Umsetzung der Modellierung von OpenQASM

Zur grafischen Darstellung von in OpenQASM implementierten Algorithmen wird das Quantum Circuit [Qua20a] Modul verwendet. Wie in Abbildung 7.6 dargestellt wird, muss der Nutzer hierzu den vollständigen OpenQASM-Code, inklusive der in Abschnitt 6.3 eingeführten Endung, einfach in den Muster-Editor einfügen. Demzufolge ist es möglich sowohl LaTeX-Quantenschaltungen als auch OpenQASM-Implementierungen im gleichen Editor einzubinden. Wenn immer ein Muster dargestellt wird, wird überprüft, ob es einen oder mehrere OpenQASM Algorithmen enthält. Ist dies der Fall, wird der Algorithmus im Frontend in Quantum Circuit importiert. Quantum Circuit konvertiert diesen anschließend in ein SVG, welches inline anstelle des OpenQASM-Codes in den dargestellten Inhalt integriert wird. Des Weiteren ist es möglich einen Link zur Darstellung des Algorithmus in Quirk [Str19] einzufügen. Mit Quirk lassen sich Quantenalgorithmen einfach modellieren und per Drag & Drop anpassen. Die aktuelle Version von Quantum Circuit bietet leider keine Möglichkeit einen Quirk-Algorithmus zu importieren. Somit ist es leider nicht möglich mit Quirk den Quantenschaltkreis anzupassen und so den OpenQASM-Code anzupassen. Da Quantum Circuit jedoch noch in Entwicklung ist, ist es möglich, dass die Quirk-Funktionalität in der Zukunft noch erweitert wird.

7.4 LaTeX-Rendering-Service

Der LaTeX-Rendering-Service wurde als eigenständiger Webservice umgesetzt, welcher LaTeX-Inhalte akzeptiert, diese rendert und in einem gewünschten Format zurück gibt. Es handelt sich dabei um ein in Java implementiertes Spring-Boot-Projekt. Da der Rendering-Service nicht ausschließlich für die Verwendung im Pattern Atlas entwickelt wird sollen mehrere Ausgabeformate unterstützt werden. LaTeX-Dokumente werden standardmäßig als PDF gerendert. Somit stellt dies das erste und

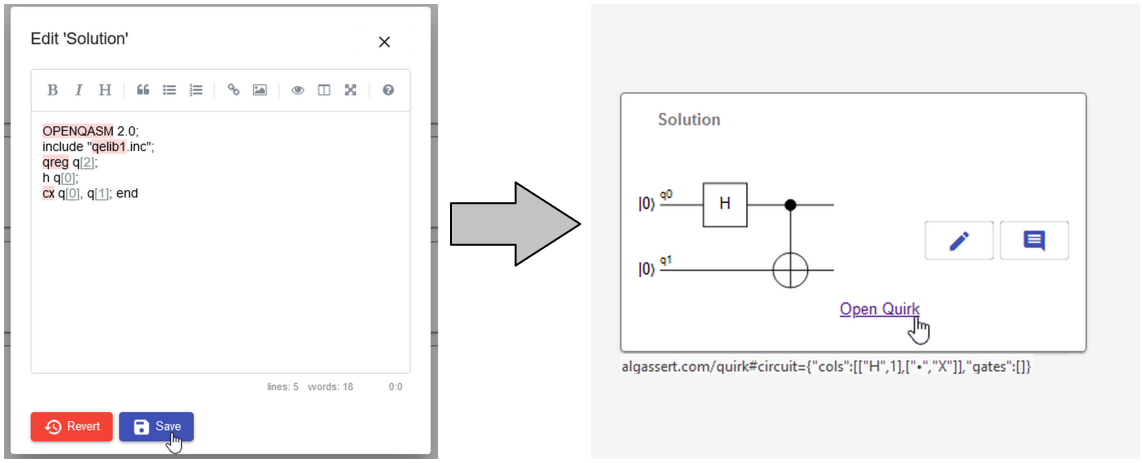


Abbildung 7.6: Ablauf der OpenQASM Modellierung

offensichtlichste Rückgabeformat dar. Da Quantenschaltkreise im Pattern Atlas als SVG dargestellt werden sollen stellt, die SVG-Rückgabe das wichtigste Rückgabeformat für dieses Projekt dar. Des Weiteren wird PNG als Rückgabeformat unterstützt.

7.4.1 Implementierung

Es ist leider nicht möglich den LaTeX-Code zur Erzeugung einer Tabelle, eines Graphen oder einer anderen Grafik einfach in ein leeres .tex Dokument einzufügen und dieses zu rendern. LaTeX hat bestimmte Restriktionen die eingehalten werden müssen, um ein Dokument erfolgreich zu rendern. So muss beispielsweise immer der Start und das Ende eines Dokuments definiert werden. Wenn der zu rendernde LaTeX-Code bestimmte Pakete verwendet, müssen diese explizit definiert werden.

Sobald der LaTeX-Rendering-Service einen Request empfängt, wird ein neues LaTeX-Dokument erstellt, welches alle im Request mitgesendeten Informationen enthält. Anschießend wird das LaTeX-Dokument mit der installierten LaTeX-Distribution gerendert. Dieser Vorgang dauert wenige Sekunden, was zu einer kurzen Wartezeit für die Ausführung eines Requests führt. Als Resultat verfügt der Rendering-Service nun über eine PDF, welches die gerenderten Inhalte enthält. Dieses wird, wenn SVG als Rückgabetyp gewählt wurde, in ein SVG konvertiert und anschließend zurückgegeben. Der LaTeX-Rendering-Service steht zur einfachen Ausführung als Docker-Container bereit.

8 Diskussion

In diesem Kapitel wird anhand eines Beispielszenarios überprüft, ob die gesetzten Ziele erreicht wurden und welche Einschränkungen gegeben sind.

8.1 Beispielszenario

Ein Nutzer will ein neues Muster erstellen. Dieses Muster soll mathematische Formeln, zwei Quantenschaltungen und eine OpenQASM-Implementierung beinhalten. Nach dem Erstellen betrachtet ein Kollege das Muster in der Detailansicht. Der Kollege entdeckt einen Fehler in einem Quantenschaltkreis und markierten diesen. Er annotiert den Bereich mit der Beschreibung des Fehlers. Der Musterautor kehrt später zu der Musterdetailansicht des zuvor erstellten Musters zurück und sieht, dass ein Quantenschaltkreis markiert wurde. Er liest den Kommentar, überprüft dessen Richtigkeit und antwortet auf den Kommentar des Kollegen. Anschließend öffnet er den Bearbeitungseditor und löst das Problem. Da es sich nur um eine kleine Anpassung handelt sollen die Kommentare erhalten bleiben, sodass sein Kollege die Antwort auf seine Bemerkung immer noch sehen kann.

1. **Erstellen eines Musters mit Quantenartefakten**, wie einer mathematischen Formel, einer in LaTeX dargestellten Quantenschaltung und einer OpenQASM Implementierung. Wie in Abbildung 8.1 zu sehen ist, können die zuvor genannten Artefakte, zwar alle in den Markdown-Editor zur Mustererstellung eingefügt werden. Das Live-Render-Feature auf der rechten Seite funktioniert jedoch nur für mathematische Formeln. Quantenschaltungen müssen extra mit dem LaTeX-Rendering-Service gerendert werden. Somit wäre es nicht sinnvoll bei jedem neu gesetzten Zeichen einen Request zum Rendern der enthaltenen Quantenschaltungen an den LaTeX-Rendering-Service zu senden. Die Quantenschaltungen werden gerendert, sobald das Muster abgespeichert wird.
2. **Anzeige des Muster in der Detailansicht**. Wie in Abbildung 8.2 dargestellt wird, kann das im vorherigen Schritt erstellte Muster nun inklusive der gerenderten Quantenschaltungen eingesehen werden.
3. **Diskussion einer Quantenschaltung**. Es ist, wie in Abbildung 8.3 dargestellt, möglich Bereiche in Quantenschaltkreisen zu markieren und diese mit Kommentaren zu annotieren. Anschließend ist es möglich die Kommentare in einer Detailansicht anzusehen und weitere Antworten auf diese zu geben.
4. **Bearbeiten eines Musters und dessen Quantenartefakte**. Im Musterbearbeitungseditor können Text, mathematische Formeln, OpenQASM Implementierungen und in LaTeX verfasste Quantenschaltungen einfach und direkt angepasst werden. Quantenschaltungen werden, im Falle einer Änderung neu gerendert. In Abbildung 8.4 wird dargestellt, wie ein

Quantikz-Artefakt aufgrund des im vorherigen Abschnitt erstellten Kommentars angepasst wird. Da es sich nur um eine kleine Anpassung und keinen neuen Quantenschaltkreis handelt, werden die Kommentare nach dem Rendern des bearbeiteten Quantenschaltkreises übernommen.

Die Durchführung des Beispielszenarios zeigt auf, dass die geplante Funktionalität erfolgreich umgesetzt wurde. Es ist mögliche mathematische Formeln, Quantenschaltungen und OpenQASM-Implementierungen darzustellen. Anschließend können diese einfach und schnell bearbeitet werden. Zum kollaborativen Arbeiten wurde eine Diskussionsfunktion hinzugefügt, mit der Kommentare zu bestimmten Bereichen präzise verfasst werden können. Lediglich die Live-Vorschau sieht unschön aus, da Qcircuit-, Quantikz- und OpenQASM-Code direkt angezeigt werden. Da es sich um eine Live-Vorschau handelt, die bei jedem neuen Zeichen angepasst wird, ist dieses Problem nicht direkt lösbar. Des Weiteren wird das OpenQASM-Modellierungsfeature wie schon in Abschnitt 6.3 erläutert wird, nicht vollständig unterstützt. So kann dieser weder einfach mittels Drag & Drop via Quirk angepasst werden noch können Implementierungen direkt via Knopfdruck in beispielsweise eine Quantencomputing-Programmierumgebung übernommen werden. Die automatische Kommentarübernahme ist in vielen Fällen praktisch, aber nicht immer optimal. Falls eine längere Verwendung des Prototyps im Mustererstellungsprozess aufzeigt, dass eine manuelle Auswahl den Nutzerwünschen besser entspricht, muss dies womöglich angepasst werden.

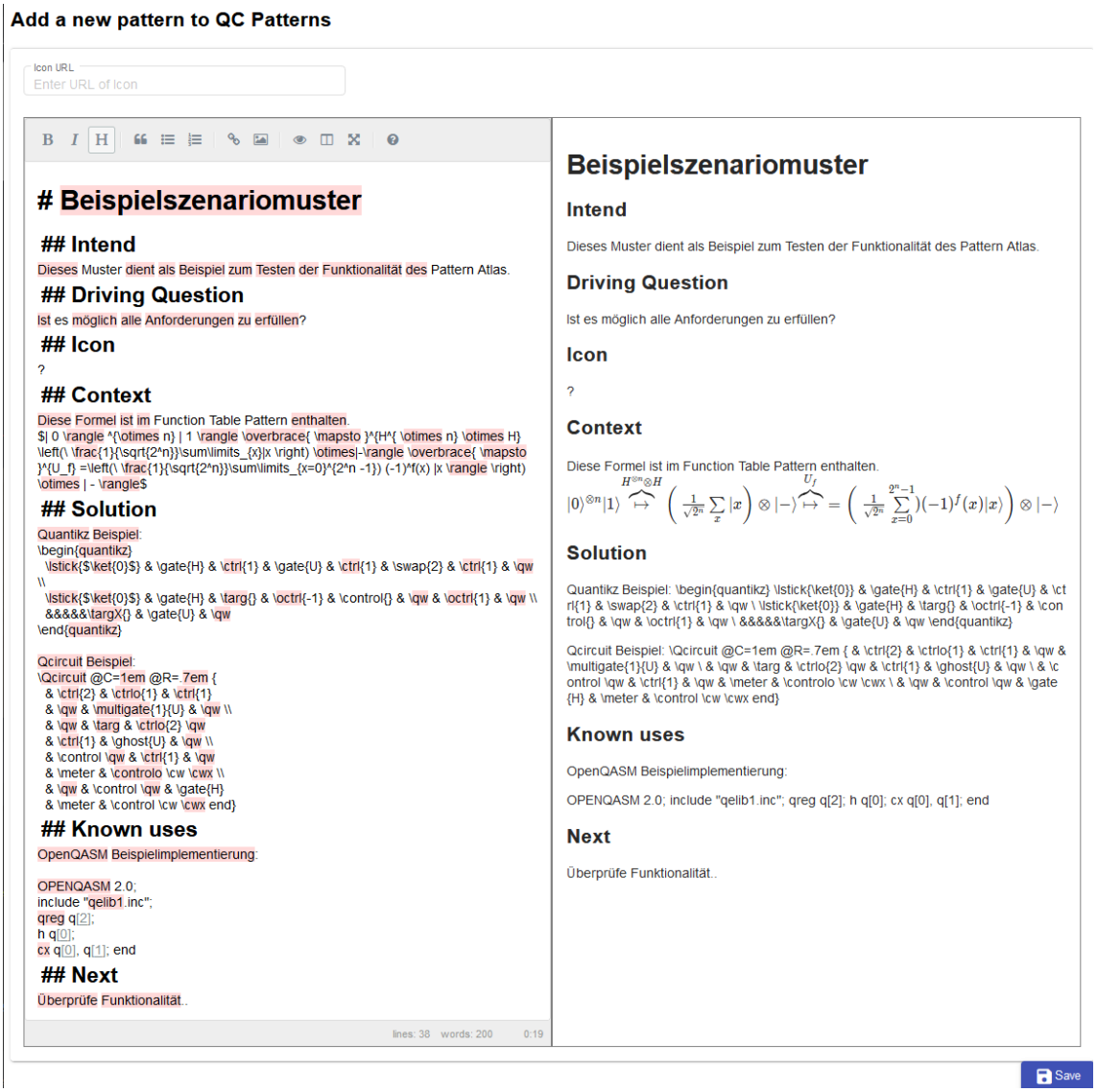


Abbildung 8.1: Erstellen eines neuen Musters im Pattern Atlas

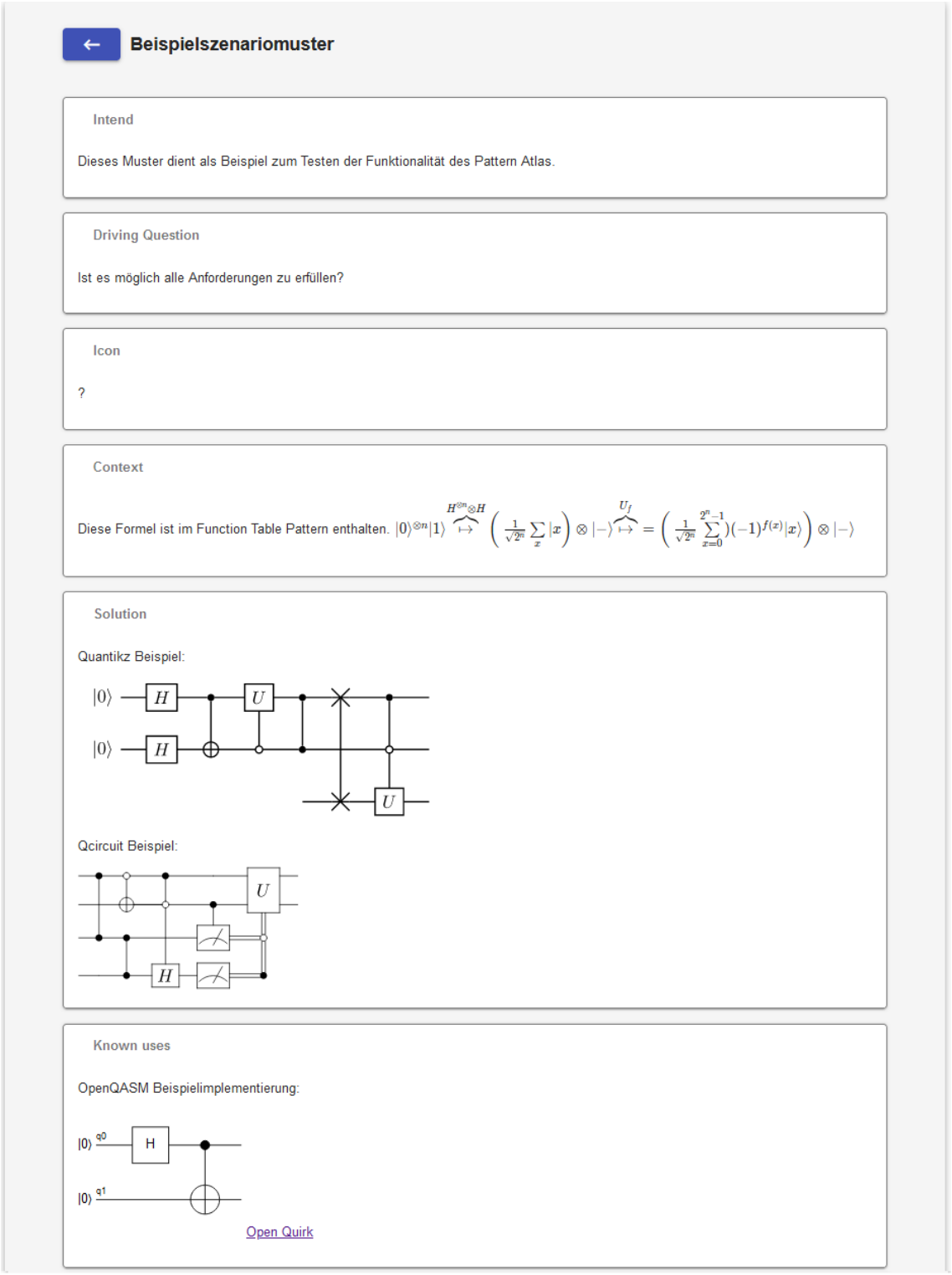


Abbildung 8.2: Detailansicht des erstellten Musters im Pattern Atlas

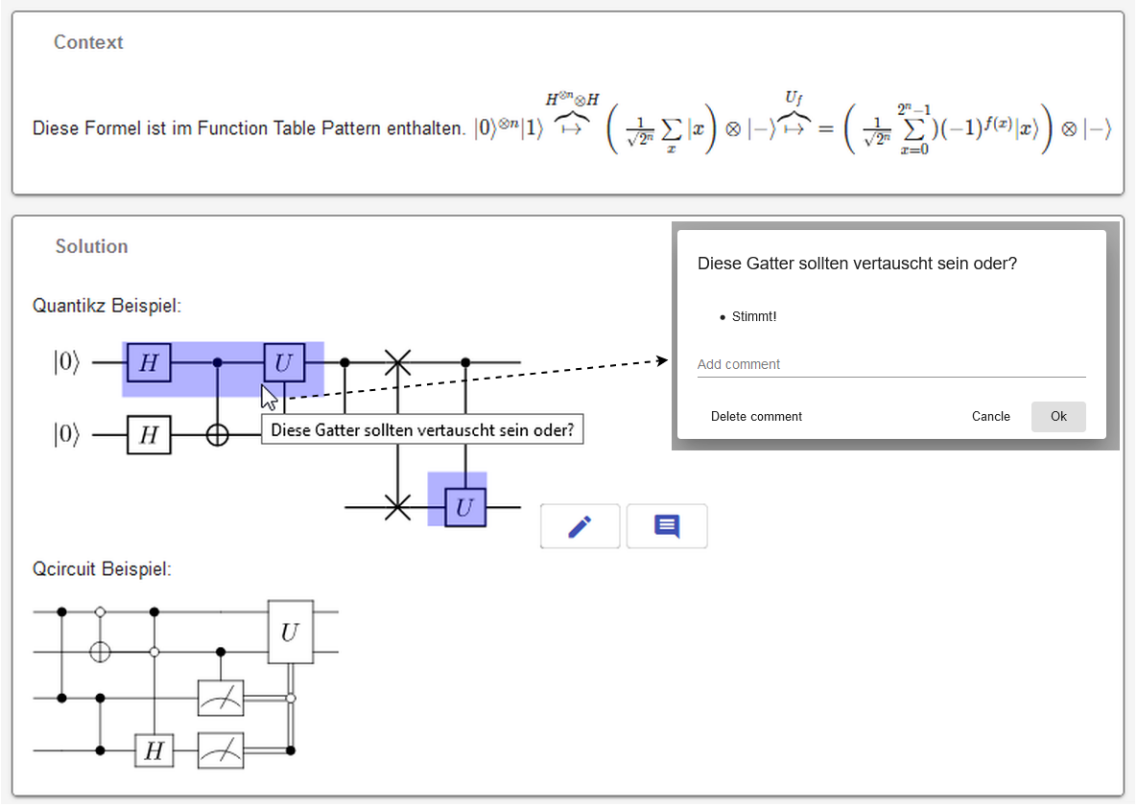


Abbildung 8.3: Diskutieren des Musters im Pattern Atlas

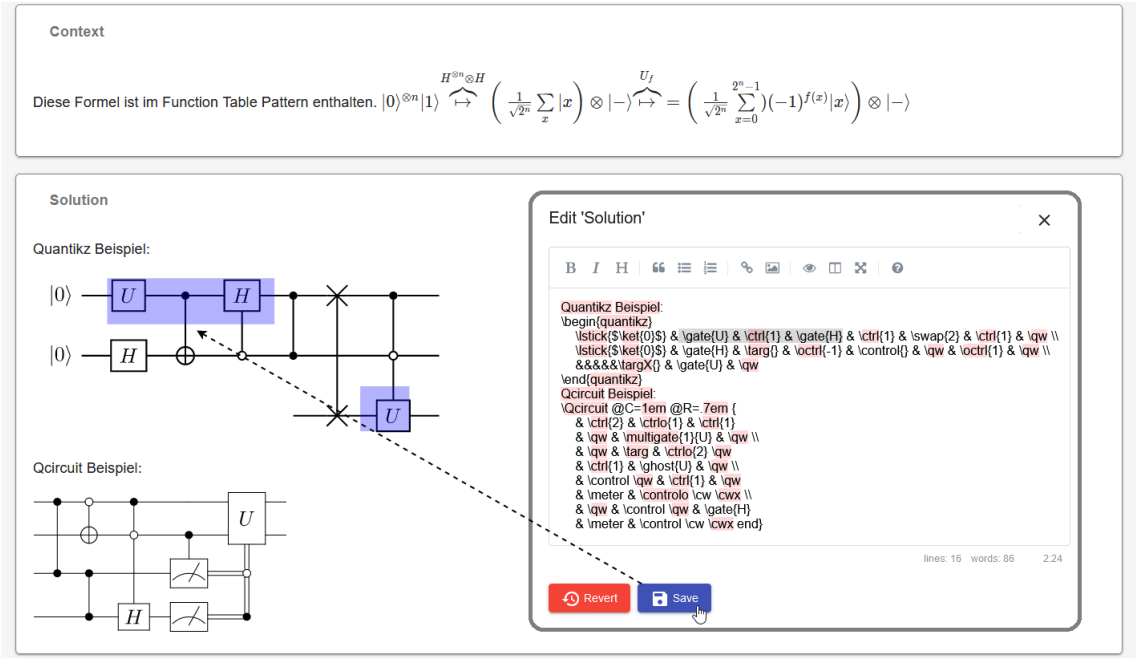


Abbildung 8.4: Editieren mit Kommentarübernahme im Pattern Atlas

9 Zusammenfassung und Ausblick

In dieser Arbeit wurde analysiert, welche spezifischen Anforderungen der Mustererstellungsprozess für Quantencomputing-Muster in einem Musterrepository erfüllen muss. Es wurde erkannt, dass Leymanns Quantencomputing-Muster [Ley19] deutlich weniger abstrakt sind, als es für Muster üblich ist. Die Quantencomputing-Muster enthalten nicht nur mathematische Formeln, sondern es sollen auch Quantenschaltungen und implementierte Algorithmen integriert werden. Die Integration aller zuvor genannten Artefakte in das Musterrepository soll gemäß der beschriebenen Richtlinien des Mustererstellungsprozesses umgesetzt werden. So sollen die Artefakte kollaborativ und einfach erstellt und bearbeitet werden können. Anhand der abgeleiteten Anforderungen wurde ein Konzept erstellt, welches schließlich als Prototyp im Pattern Atlas umgesetzt wurde. Mit dem Prototyp ist es möglich in LaTeX verfasste mathematische Formeln und Quantenschaltungen darzustellen und in OpenQASM implementierte Quantenalgorithmen zu integrieren. Diese Artefakte können einfach in einem Markdown-Editor bearbeitet werden. Um den kollaborativen Mustererstellungsprozess weiter zu unterstützen, ist es möglich ausgewählte Bereiche der grafisch dargestellten Quantenschaltungen zu markieren und zu kommentieren. Andere Nutzer können auf Kommentare eingehen und auf diese antworten. Somit bietet der Prototyp eine Diskussionsplattform für grafisch dargestellte Quantenschaltungen.

Um die Funktionalität des Prototyps in Zukunft weiter zu verbessern und den Mustererstellungsprozess noch angenehmer und effizienter zu gestalten, wurden folgende Erweiterungsoptionen identifiziert:

- Um den Übergang zwischen Mustererstellungs- und Musteranwendungsphase noch fließender zu gestalten, wäre es möglich eine Option anzubieten, mit welcher die OpenQASM-Implementierungen lokal gespeichert, oder direkt in eine Quantencomputing-Programmierumgebung exportiert werden können. Selbiges kann für den Import umgesetzt werden. Momentan werden OpenQASM-Implementierungen direkt in den Mustertext integriert. Um eine größere Zahl von OpenQASM-Implementierungen zu integrieren, würde sich eine Dateiimportoption anbieten.
- Da die Quirk-Integration nicht vollständig umgesetzt werden konnte, wäre es möglich zu untersuchen ob eine Quirk zu OpenQASM-Transformation möglich ist, sodass OpenQASM-Algorithmen mit Quirk modelliert und simuliert werden können und getroffene Änderungen anschließend automatisch in OpenQASM übernommen werden.
- OpenQASM-Algorithmen werden momentan von Quantum Circuit direkt im Frontend gerendert. Dies führt dazu, dass die Quantenschaltungen nicht im Backend gespeichert werden und somit auch keine Kommentare mit ihnen assoziiert werden können. Um Kommentare und Diskussionen auch für OpenQASM-Algorithmen zu ermöglichen, müsste eine Anpassung des Renderprozesses der OpenQASM-Quantenschaltungen vorgenommen werden.

Literaturverzeichnis

[Aca20] K. Academy. *KaTeX*. 2020. URL: https://katex.org/ (zitiert auf S. 31, 39).

[AG20] M. Amy, V. Gheorghiu. „staq-A full-stack quantum processing toolkit“. In: *Quantum Science and Technology* (2020) (zitiert auf S. 20).

[AIS77] C. Alexander, S. Ishikawa, M. Silverstein. *A Pattern Language: Towns, Buildings, Construction*. 1977 (zitiert auf S. 15).

[AR05] R. Apfelbacher, A. Rozinat. *Compositional Structures*. 2005. URL: http://www.fmc-modeling.org/download/notation_reference/Reference_Sheet-Block_Diagram.pdf (zitiert auf S. 33).

[Bel17] A. Bellamy-Royds. *How to Scale SVG*. 2017. URL: https://css-tricks.com/scale-svg/ (zitiert auf S. 20).

[Bos20] M. Bostock. *Data-Driven Documents*. 2020. URL: https://d3js.org/ (zitiert auf S. 35).

[Bry20] S. T. F. Bryan Eastin. *Q-circuit Tutorial*. 2020. URL: http://physics.unm.edu/CQuIC/Qcircuit/Qtutorial.pdf (zitiert auf S. 26, 31, 33).

[CBSG17] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta. „Open quantum assembly language“. In: *arXiv preprint arXiv:1707.03429* (2017) (zitiert auf S. 20).

[CEP+18] P. J. Coles, S. Eidenbenz, S. Pakin, A. Adedoyin, J. Ambrosiano, P. Anisimov, W. Casper, G. Chennupati, C. Coffrin, H. Djidjev et al. „Quantum algorithm implementations for beginners“. In: *arXiv* (2018), arXiv–1804 (zitiert auf S. 19).

[Cro05] A. Cross. *Qasm-Tools*. 2005. URL: https://www.media.mit.edu/quanta/quanta-web/projects/qasm-tools/ (zitiert auf S. 20).

[CTA20] CTAN. *What are TeX and its friends?* 2020. URL: https://www.ctan.org/tex (zitiert auf S. 13, 26).

[Deb19] D. Debroy. *Tools for creating quantum circuit diagrams*. 2019. URL: https://quantumcomputing.stackexchange.com/questions/4580/tools-for-creating-quantum-circuit-diagrams (zitiert auf S. 26).

[DKT05] J. Deng, E. Kemp, E. G. Todd. „Managing UI pattern collections“. In: *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*. 2005, S. 31–38 (zitiert auf S. 21).

[El 20] T. El Dandachi. *qasm-circuit-preview*. 2020. URL: https://github.com/tareqdandachi/qasm-circuit-preview (zitiert auf S. 29, 30).

[FBBL15] C. Fehling, J. Barzen, U. Breitenbücher, F. Leymann. „A Process for Pattern Identification, Authoring, and Application“. Deutsch. In: *Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP)*. ACM, Jan. 2015, S. 1–9. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2015-50%5C&engl=0 (zitiert auf S. 13, 15–18, 25, 26).

[FBFL14] C. Fehling, J. Barzen, M. Falkenthal, F. Leymann. „PatternPedia-collaborative pattern identification and authoring“. In: *Proceedings of PURPLSOC (Pursuit of Pattern Languages for Societal Change). The Workshop*. 2014, S. 252–284 (zitiert auf S. 21).

[FL17] M. Falkenthal, F. Leymann. „Easing Pattern Application by Means of Solution Languages“. In: *roceedings of the 9th International Conference on Pervasive Patterns and Applications*. 2017 (zitiert auf S. 23).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer, 2014. DOI: 10.1007/978-3-7091-1568-8 (zitiert auf S. 13, 15, 21).

[Gas18] R. Gast. *Europas Antwort auf den Quantencomputer-Hype*. Okt. 2018. URL: https://www.spektrum.de/news/europas-antwort-auf-den-quantencomputer-hype/1605166 (zitiert auf S. 13).

[Gau07] A. Gaudeul. „Do open source developers respond to competition? The LaTeX case study“. In: *Review of Network Economics* 6.2 (2007) (zitiert auf S. 26).

[GCBA20] J. Gambetta, J. Chow, E. Bäumer, A. Asfaw. *IBM Quantum Challenge results: Billions and billions of circuits*. 2020. URL: https://www.ibm.com/blogs/research/2020/05/quantum-challenge-results/ (zitiert auf S. 26).

[GGA19] S. Garhwal, M. Ghorani, A. Ahmad. „Quantum Programming Language: A Systematic Review of Research Topic and Top Cited Languages“. In: *Archives of Computational Methods in Engineering* (2019), S. 1–22 (zitiert auf S. 26, 27).

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994 (zitiert auf S. 15).

[Gid19] E. M. Gidney Craig. *How a quantum computer could break 2048-bit RSA encryption in 8 hours*. Mai 2019. URL: https://www.technologyreview.com/s/613596/how-a-quantum-computer-could-break-2048-bit-rsa-encryption-in-8-hours/ (zitiert auf S. 13).

[Goo20] Google. *Angular*. 2020. URL: https://angular.io/ (zitiert auf S. 32).

[Gro20] P. G. D. Group. *PostgreSQL*. 2020. URL: https://www.postgresql.org/ (zitiert auf S. 32).

[IAA20] IAAS. *Pattern Atlas*. 2020. URL: https://github.com/PatternAtlas (zitiert auf S. 13, 21, 23).

[IBM20a] IBM. *OpenQASM*. 2020. URL: https://github.com/Qiskit/openqasm (zitiert auf S. 20, 26).

[IBM20b] IBM. *Qiskit*. 2020. URL: https://qiskit.org/ (zitiert auf S. 26, 29, 30).

[IBM20c] IBM. *What is Quantum Compuing*. 2020. URL: https://www.ibm.com/quantum-computing/learn/what-is-quantum-computing (zitiert auf S. 19).

[Int20] IntMath.com. *KaTeX and MathJax Comparison Demo*. 2020. URL: https://www.intmath.com/cg5/katex-mathjax-comparison.php (zitiert auf S. 31).

[IS15] P. S. Inventado, P. Scupelli. „Towards an open, collaborative repository for online learning system design patterns“. In: *eLearning Papers* 42 (2015), S. 2 (zitiert auf S. 22).

[Jac17] M. Jackson. *6 Things Quantum Computers Will Be Incredibly Useful For*. 2017. URL: https://singularityhub.com/2017/06/25/6-things-quantum-computers-will-be-incredibly-useful-for/ (zitiert auf S. 19).

[Kan98] B. E. Kane. „A silicon-based nuclear spin quantum computer“. In: *nature* 393.6681 (1998), S. 133–137 (zitiert auf S. 13).

[KGA+18] N. Khammassi, G. G. Guerreschi, I. Ashraf, J. W. Hogaboam, C. G. Almudever, K. Bertels. „cqasm v1. 0: Towards a common quantum assembly language“. In: *arXiv preprint arXiv:1805.09607* (2018) (zitiert auf S. 20).

[KISV16] C. Köppe, P. S. Inventado, P. Scupelli, U. Van Heesch. „Towards extending online pattern repositories: supporting the design pattern lifecycle“. In: *Proceedings of the 23rd Conference on Pattern Languages of Programs*. 2016, S. 1–26 (zitiert auf S. 21).

[KLM+07] P. Kaye, R. Laflamme, M. Mosca et al. *An introduction to quantum computing*. Oxford university press, 2007 (zitiert auf S. 19).

[Kos19] S. Kosub. „A note on the triangle inequality for the Jaccard distance“. In: *Pattern Recognition Letters* 120 (2019), S. 36–38 (zitiert auf S. 45).

[Ley19] F. Leymann. „Towards a Pattern Language for Quantum Algorithms“. In: *International Workshop on Quantum Technology and Optimization Problems*. Springer. 2019, S. 218–230 (zitiert auf S. 13, 15, 19–21, 23, 25, 26, 31, 40, 53).

[Ma18] E. Ma. *3 basic Distance Measurement in Text Mining*. 2018. URL: https://towardsdatascience.com/3-basic-distance-measurement-in-text-mining-5852becff1d7 (zitiert auf S. 45).

[Mel20] U. of Melbourne. *QUI*. 2020. URL: https://qui.science.unimelb.edu.au/ (zitiert auf S. 27, 28).

[Mic16] Microsoft. *SVG Coordinate Transformations*. 2016. URL: https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/samples/hh535760(v=vs.85)?redirectedfrom=MSDN (zitiert auf S. 43, 44).

[Mic19] MicroDroid. *Upgrade KaTeX to 0.11.1*. 2019. URL: https://github.com/waylonflinn/markdown-it-katex/pull/29 (zitiert auf S. 40).

[Moz20a] Mozilla. *Adding vector graphics to the Web*. 2020. URL: https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Adding_vector_graphics_to_the_Web (zitiert auf S. 35).

[Moz20b] Mozilla. *SVG: SCalable Vector Graphics*. 2020. URL: https://developer.mozilla.org/en-US/docs/Web/SVG (zitiert auf S. 20).

[Num20] NumFOCUS. *MathJax*. 2020. URL: https://www.mathjax.org/ (zitiert auf S. 31).

[pan20] pandoc. *Pandoc User's Guide*. 2020. URL: https://pandoc.org/MANUAL.html#math (zitiert auf S. 39).

[Pre18] J. Preskill. „Quantum Computing in the NISQ era and beyond“. In: *Quantum* 2 (Aug. 2018), S. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: https://doi.org/10.22331/q-2018-08-06-79 (zitiert auf S. 19).

[Qua19] T. on the Quantikz PackageAlastair Kay. *Tutorial on the Quantikz Package*. 2019. URL: https://arxiv.org/pdf/1809.03842.pdf (zitiert auf S. 26, 31, 33).

[Qua20a] Quantastica. *Quantum Circuit Simulator*. 2020. URL: https://www.npmjs.com/package/quantum-circuit (zitiert auf S. 28–30, 35, 45).

[Qua20b] Quantiki. *List of QC simulators*. 2020. URL: https://www.quantiki.org/wiki/list-qc-simulators (zitiert auf S. 27).

[QuT20] QuTech. *Quantum Inspire*. 2020. URL: https://www.quantum-inspire.com/ (zitiert auf S. 27, 28).

[RFJZ15] R. Reiners, M. Falkenthal, D. Jugel, A. Zimmermann. „Requirements for a collaborative formulation process of evolutionary patterns“. In: *Proceedings of the 18th European Conference on Pattern Languages of Program*. 2015, S. 1–12 (zitiert auf S. 21).

[Roe18] J. Roell. *Demystifying Quantum Gates — One Qubit At A Time*. 2018. URL: https://towardsdatascience.com/demystifying-quantum-gates-one-qubit-at-a-time-54404ed80640 (zitiert auf S. 19).

[SDC+20] S. Sivarajah, S. Dilkes, A. Cowtan, W. S. A. Edgington, R. Duncan. „t | ket: A Retargetable Compiler for NISQ Devices“. In: *arXiv preprint arXiv:2003.10611* (2020) (zitiert auf S. 20, 26).

[Sof19] C. Software. *ISO/IEC 25010*. 2019. URL: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?limit=3%5C&limitstart=0 (zitiert auf S. 30).

[Sta20] StackExchange. *Quantum Computing Stack Exchange*. 2020. URL: https://quantumcomputing.stackexchange.com/ (zitiert auf S. 22).

[Str19] Strilanc. *Quirk*. 2019. URL: https://github.com/Strilanc/Quirk (zitiert auf S. 28, 29, 45).

[Stu20] U. Stuttgart. *qc-atlas*. 2020. URL: https://github.com/UST-QuAntiL/qc-atlas (zitiert auf S. 36).

[Sua18] A. Suau. *qasm2image*. 2018. URL: https://pypi.org/project/qasm2image/ (zitiert auf S. 29, 30).

[tex18] texfaq. *Math on the Web*. 2018. URL: https://www.texfaq.org/FAQ-mathml (zitiert auf S. 31).

[unb14] unbekannt. *Quantum Circuit Inspector JSFiddle*. 2014. URL: https://jsfiddle.net/c4f5z73v/2/ (zitiert auf S. 27, 28).

[VMw20] VMware. *Spring Boot*. 2020. URL: https://spring.io/projects/spring-boot (zitiert auf S. 32).

[way16] waylonflinn. *Markdown-it-katex*. 2016. URL: https://github.com/waylonflinn/markdown-it-katex (zitiert auf S. 39).

[wyb17] wybiral. *Quantum Circuit Simulator*. 2017. URL: https://github.com/qcsimulator/qcsimulator.github.io (zitiert auf S. 28).

[Xan20] Xanadu. *strawberryFieldsInteractive*. 2020. URL: https://strawberryfields.ai/ (zitiert auf S. 27, 28).

[ZLC00] X. Zhou, D. W. Leung, I. L. Chuang. „Methodology for quantum logic gate construction“. In: *Physical Review A* 62.5 (2000), S. 052316 (zitiert auf S. 23, 27).

Alle URLs wurden zuletzt am 28. 09. 2020 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