Institut für Architektur von Anwendungssystemen

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Entwicklung eines Frameworks für die Nutzung von konkreten Lösungen der Quantencomputing Mustersprache

Ahmed Ebrahim Aldekal

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Dr. h. c. Frank Leymann
Betreuer/in:	Daniel Georg M.Sc., Daniel Vietz M.Sc.
Beginn am:	12. Juli 2023
Beendet am:	19. Januar 2024

Kurzfassung

Muster finden in vielfältigen Bereichen der Informationstechnologie (IT) Anwendung und bieten abstrakte, bewährte Lösungsansätze für wiederkehrende Probleme. Insbesondere im Quantencomputing (QC) sind Muster von Bedeutung, da sie dazu dienen, QC-Algorithmen zu beschreiben und deren Implementierung zu erleichtern. Jedoch besteht das Problem des Mangels an Methoden zur effektiven Integration konkreter Lösungen in ein Gesamtkonzept. Zusätzlich mangelt es an einer angemessenen Speicher- und Wiederverwendungsstrategie für diese Lösungen, was zum Verlust wertvollen Wissens führt. Diese Arbeit stellt ein Framework vor, welches sich auf die Speicherung sowie Kombination konkreter Lösungen konzentriert und anhand der QC-Mustersprache veranschaulicht wird. Das Framework integriert und erweitert verschiedene Open-Source-Anwendungen zur Muster-Verwaltung. Die Evaluation des Frameworks wird anhand eines spezifischen Anwendungsszenarios durchgeführt, bei dem der Deutsch-Algorithmus mittels der Muster und deren konkreten Lösungen realisiert wird. Das vorgestellte Framework erleichtert die Zusammenführung verschiedener Muster zur Entwicklung von QC-Algorithmen. Es vereinfacht den Einsatz der Mustersprache und unterstützt somit den Einstieg in das QC. Besonders technische Feinheiten und quantenspezifische Details, die häufig Hürden darstellen, sollen durch dieses Framework für den Anwender in der Zukunft vereinfacht werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Verzeichnis der Algorithmen

Abkürzungsverzeichnis

API Application Programming Interfaces. 31

HTTP Hypertext Transfer Protocol. 31

IT Informationstechnologie. 3

JPA Spring Data JPA (Java Persistence API). 31

OpenQASM-2 Open Quantum Assembly Language 2. 25

QC Quantencomputing. 3

SQL Structured Query Language. 31

UI User Interface. 18

1 Einleitung

QC steht an der Schwelle zu einer revolutionären Technologie, die das Potenzial hat, die Art und Weise, wie wir komplexe Probleme in verschiedenen Bereichen lösen, grundlegend zu verändern [LJL+10]. Dieses vielversprechende Feld der Informatik birgt allerdings auch Herausforderungen, insbesondere in Bezug auf seine Programmierung [Ley19]. Die Komplexität von Quantenalgorithmen und die Eigenheiten von Quantencomputern machen es notwendig, neue Methoden und Richtlinien zu entwickeln, um diese Technologie effektiv nutzen zu können.

In diesem Kontext spielen Muster eine entscheidende Rolle. Muster sind bewährte Lösungen für wiederkehrende Probleme. Sie dokumentieren Fachwissen und bewährte Vorgehensweisen, die sich in bestimmten Situationen als effektiv erwiesen haben. Muster bieten einen strukturierten Ansatz zur Problemlösung und dienen als Leitfaden, um ähnliche Probleme zu identifizieren und passende Lösungen anzuwenden [FL17][FBBL17]. Ursprünglich wurden die ersten Muster von Christopher Alexander et al. [Ale77] für die Architektur von Städten und Gebäuden entwickelt. Ihre Wirksamkeit und Vielseitigkeit haben dazu geführt, dass Muster in vielen Bereichen, einschließlich der Informatik, Anwendung finden. So wurden Muster in der Mensch-Computer-Interaktion, zur Integration von Unternehmensanwendungen [HW02] und zur Unterstützung des objektorientierten Designs [GHJV94] erfolgreich eingesetzt. Im Bereich des Quantencomputing stellen Muster eine unerlässliche Ressource dar, um die Komplexität der Technologie zu beherrschen und ihre Anwendung zu erleichtern. Die Entwicklung einer QC-Mustersprache, basierend auf der grundlegenden Arbeit von Leymann [Ley19], ermöglicht es uns, Richtlinien und bewährte Verfahren für das Design und die Implementierung von Quantencomputing-Lösungen zu etablieren.

In den folgenden Abschnitten wird der Einsatz von Mustern näher beleuchtet. Abschnitt 1.1 diskutiert die Herausforderungen bei der Musterentwicklung und deren Abstraktion. Abschnitt 1.2 fokussiert auf die Notwendigkeit, ein Framework zur Unterstützung der Anwendung dieser Muster zu entwickeln. Schließlich gibt Abschnitt 1.3 einen Überblick über den Aufbau der Arbeit.

1.1 Motivation

Das Muster- bzw. der Mustersprachenerstellungsvorgang wird in der Regel manuell von Experten der jeweiligen Domäne durchgeführt [Rei12] [FBBL15]. Dabei werden bewährte Lösungen für wiederkehrende Probleme in einem bestimmten Bereich identfiziert und abstrahiert. Die abstrahierte Lösung sollte eher generisch als spezifisch und insbesondere technologieunabhängig sein, sodass sie auf vielfältige Arten implementiert werden kann [Rei12]. Die Abstraktion einer Lösung (bzw. Mustererstellung) ermöglicht es, eine größere Klasse von Problemen abzudecken. Allerdings geht dabei auch Wissen verloren, was bedeutet, dass die Anwendung von Mustern für spezifische Anwendungsfälle schwieriger wird, da konkrete Lösungen, d.h. Implementierungen eines Musters, während des Erstellungsprozesses verloren gehen [FL17].

Insgesamt lässt sich feststellen, dass das Konzept der Muster zwar auf Verallgemeinerung und Abstraktion abzielt, jedoch oft Schwierigkeiten bei der Anwendung des abstrahierten Wissens auf konkrete Probleme auftreten können [FBB+14].

Alexander et al. [Ale77] führten den Begriff „Mustersprache“ ein, um zu betonen, dass Muster in der Regel nicht isolierte Wissensfragmente sind, sondern in Kombination verwendet werden. Die Muster können miteinander verknüpft werden, um Mustersprachen zu bilden, und so die Navigation durch die Domäne zu erleichtern [FL17] [FBBL17]. Dadurch können Muster auf verschiedenen Abstraktionsebenen in Mustersprachen organisiert werden [FL17][FBBL17] [FBB+16]. Jedoch reicht allein die Mustersprache nicht für das musterbasierte Problemlösen aus; es bedarf auch einer Lösungssprache und Navigationsmöglichkeiten innerhalb dieser Sprache. Durch das Fehlen von solchen Navigationsmöglichkeiten wird die Wiederverwendung von vorhandenen konkreten Lösungen erschwert, insbesondere wenn es viele verschiedene und technologiespezifische Implementierungen von Mustern gibt. Infolgedessen ist es schwierig, konkrete Lösungen zu kombinieren, um eine umfassende Lösung zu realisieren. In dieser Arbeit wird untersucht, wie diese Wissenslücke zwischen der abstrakten Beschreibung und den konkreten Lösungen im Bereich des QC geschlossen werden kann.

1.2 Ziele

Das Ziel dieser Bachelorarbeit besteht darin, ein Framework zu entwickeln, das Benutzer bei der Verwendung von QC Mustern unterstützt. Dazu sollen relevante Muster und ihre konkreten Lösungen über eine User Interface (UI) identifiziert werden. Anschließend wird eine Gesamtlösung (Aggregation) vorgeschlagen, die diese Muster umfasst. Die Erstellung der Gesamtlösung mit Hilfe der Muster wird durch eine grafische Benutzerschnittstelle ermöglicht.

Diese Arbeit verfolgt das Ziel, die Integration und Navigation von konkreten Implementierungen von QC-Mustern zu unterstützen und die Aggregation einer umfassenden Gesamtlösung zu ermöglichen. Um dieses Ziel zu erreichen, umfasst der Ansatz mehrere Schlüsselaufgaben: Dazu gehört die Analyse von Metainformationen zur Abbildung konkreter Lösungen im „Pattern Atlas“ [Pla] und die Erweiterung dieses Atlases durch Verlinkung der spezifischen Lösungen. Ein weiterer wichtiger Schritt ist die Implementierung und Dokumentation einer Schnittstelle zur Anbindung an die Winery [Con23], welche als grafische Benutzeroberfläche für die Auswahl von Mustern dient. Diese gezielten Maßnahmen ermöglichen es, die Lösungen der ausgewählten Muster zu einer kohärenten Gesamtlösung zu aggregieren. Abschließend ist die Umsetzung eines Szenarios zur Validierung und Nutzung dieser QC-Muster von entscheidender Bedeutung, um die Praxistauglichkeit und Effizienz des vorgestellten Ansatzes zu demonstrieren und weiter zu optimieren.

1.3 Gliederung

Die Arbeit ist wie folgt gegliedert:

Das Kapitel 2 – Grundlagen und verwandte Arbeiten beschreibt und erläutert die thematischen Grundlagen, die für diese Arbeit relevant sind. Dabei wird ein umfassender Überblick über Muster, Mustersprachen, Lösungssprachen und Musterimplementierungen gegeben.

Darüber hinaus wird in diesem Kapitel auch die QC-Mustersprache behandelt. Es werden auch Aspekte wie die Konkretisierung von Mustern und die Aggregation von Musterimplementierungen betrachtet. Hierbei werden relevante Arbeiten aus demselben Themengebiet als Grundlage herangezogen.

Das Kapitel 3 – Problemstellung und Anforderungen legt die spezifische Problemstellung der Arbeit dar. Es werden die Herausforderungen und Fragen erläutert. Zudem werden die Anforderungen definiert, die für die Entwicklung des Frameworks und die Lösung des Problems relevant sind.

Das Kapitel 4 – Konzept zeigt den Architekturentwurf des Frameworks auf.

Das Kapitel 5 – Implementierung beschreibt die praktische Umsetzung des Frameworks, einschließlich technischer Details und Implementierungsschritte.

Das Kapitel 6 – Anwendungsszenario präsentiert ein konkretes Anwendungsszenario für das Framework, um zu zeigen, wie es in der Praxis eingesetzt werden kann.

Das Kapitel 7 – Evaluation beschreibt, wie das Framework getestet und bewertet wurde, einschließlich der Methodik und der Ergebnisse der Evaluation.

Das Kapitel 8 – Zusammenfassung und Ausblick fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

2 Grundlagen und verwandte Arbeiten

In diesem Kapitel werden die Begriffe und Konzepte aus dem Bereich der Mustersprachen und QC, die für das Verständnis der Arbeit notwendig sind, erläutert.

2.1 Muster und Mustersprachen

Im Kontext der IT und der Informatik beschreiben Muster wiederkehrende Strukturen, Lösungsansätze oder bewährte Vorgehensweisen, die zur Lösung bestimmter Probleme oder zur Entwicklung von Anwendungen eingesetzt werden können. Solche Muster dienen als abstrakte Modelle, die helfen, wiederkehrende Herausforderungen zu identifizieren und effiziente Lösungen bereitzustellen [FBB+14][FBB+16][FL17][FBBL17]. Ursprünglich entwickelten Alexander et al. [Ale77] die ersten Muster für die Architektur von Städten bzw. Gebäuden. Dieses Konzept hat sich mittlerweile in verschiedenen Anwendungsbereichen der Informatik etabliert, wie zum Beispiel in den *Enterprise Integration Patterns* [HW02].

Grundsätzlich lässt sich festhalten, dass Muster menschenlesbare Artefakte sind, die Problembeschreibungen mit abstraktem Lösungswissen kombinieren [FBB+14]. Sie sind technologieunabhängige Beschreibungen und Vorgehensweisen, die für die Lösung des definierten Problems notwendig sind. Durch ihre Abstraktion ermöglichen Muster, dass bewährte Lösungen, die von erfahrenen Experten abgeleitet wurden, weitergegeben und von anderen verstanden und angewendet werden können [HW02].

Fehling et al. [FBBL15] präsentierten für die Domänen in der Informatik ein interaktives Verfahren, das sich auf drei Kernbereiche konzentriert: (i) Identifizierung von Mustern, (ii) Erstellung von Mustern und (iii) Anwendung von Mustern.

Muster weisen üblicherweise gemeinsame Attribute auf, die jedoch je nach Fachgebiet variieren können [Ley19]. Um eine einheitliche Struktur zu gewährleisten, haben Alexander et al. [Ale77] ein Schema definiert, das für alle Muster gleichermaßen gilt. Gemäß diesem Schema folgt jedes Muster einem festgelegten Aufbau: Zu Beginn wird ein Bild bzw. Icon präsentiert, das das Muster visuell veranschaulicht. Das Bild bzw. Icon ist ein Mnemonic, mit dem das Muster wiedererkannt werden kann. Anschließend folgt ein Absatz, der den Kontext des Problems umreißt, indem er den Rahmen und die spezifischen Umstände beschreibt, in denen das Muster relevant ist. Im Anschluss wird der Titel des Musters genannt, der prägnant und aussagekräftig sein sollte, um das Muster schnell identifizieren zu können. Daraufhin folgt der Kontext, der in der Regel den längsten Abschnitt des Musters bildet. Hier wird das zugrundeliegende Problem detailliert beschrieben. Alexander et al. gehen in diesem Abschnitt auch auf den empirischen Hintergrund ein, belegen die Gültigkeit des Problems und stellen verschiedene Variationen oder Ausprägungen der Problemstellung vor. Nachdem das Problem ausführlich erläutert wurde, wird die Lösung präsentiert. Diese Lösung ist in ihrer Abstraktheit konzipiert, um wiederholt auf verschiedene Kontexte angewandt werden zu können.

Es ist jedoch wichtig anzumerken, dass die Anwendung der Lösung nicht zwangsläufig zu identischen Umsetzungen führen muss. Es besteht Raum für individuelle Anpassungen und Interpretationen. Abschließend werden im Muster Referenzen auf andere Muster angegeben. Diese Verweise dienen dazu, die Verbindung zwischen verschiedenen Mustern herzustellen und ermöglichen es dem Leser, weitere Quellen zu konsultieren und sich eingehender mit dem Thema auseinanderzusetzen, da viele Muster meist zusammen mit anderen Mustern verwendet werden.

Während viele Muster einem ähnlichen Schema folgen, gibt es kein allgemeines Format, das für alle Domänen geeignet ist [FBBL15]. Beispielsweise beschreiben Gamma et al. [GHJV94] anhand einer festen Struktur einen Erstellvorgang bzw. die Dokumentation von Mustern. Nach ihnen besteht ein Muster im Allgemeinen aus vier wesentlichen Elementen. Diese Elemente umfassen den Namen des Musters, das Problem, das gelöst werden soll, die Lösung und die Konsequenzen. Der Name dient als kurze und prägnante Beschreibung des Designproblems und seiner Lösungen. Das Problem definiert den Kontext, in dem das Muster angewendet werden sollte und erklärt die Herausforderung ausführlicher. Die Lösung beschreibt die Designelemente und ihre Beziehungen, Verantwortlichkeiten und Zusammenarbeit. Schließlich werden die Konsequenzen diskutiert, die sich aus der Anwendung des Musters ergeben können.

In seiner Arbeit [Ley19] fasst Leymann die gemeinsamen Attribute eines Musters zusammen, die unabhängig vom Bereich sind. Diese Attribute umfassen:

Name
: Dient zur Identifizierung des zugrunde liegenden Problems.

Zielsetzung
: Formuliert prägnant das angestrebte Ziel der beschriebenen Lösung.

Symbol
: Erlaubt eine visuelle Erfassung des Musters. Die symbolische Darstellung kann je nach Modell und Kontext variieren.

Problembeschreibung
: Zusammenfassung des zu lösenden Problems in präziser Form, um dem Leser eine schnelle Beurteilung der Relevanz des Musters für sein spezifisches Problem zu ermöglichen.

Kontext
: Beschreibt die Situation, die zur Entstehung des Problems geführt hat.

Lösung
: Zentrales Element des Musters. Bietet eine abstrakte Beschreibung, wie das Problem gelöst werden kann. Die Problembeschreibung und die Lösung bilden zusammen das Herzstück des Musters. In einigen Fällen können auch verschiedene Lösungsansätze in Abhängigkeit vom Kontext erwähnt werden.

Bekannte Anwendungen
: Verweise auf bereits existierende Algorithmen oder Anwendungen, die das Muster nutzen. Dadurch wird belegt, dass das Problem wiederkehrend ist und die vorgeschlagene Lösung erprobt wurde.

Referenzen zu anderen Mustern
: Schaffen Verbindungen zu verwandten Mustern und bieten dem Leser die Möglichkeit, sich weiter mit dem Thema auseinanderzusetzen und auf zusätzliche Quellen zurückzugreifen.

Der letzte Punkt ist von besonderer Bedeutung, da Muster häufig miteinander verknüpft sind. Diese Verknüpfungen bilden ein navigierbares Netzwerk von Mustern, das als Mustersprache bezeichnet wird [Ale77][FBB+14][FBBL15]. In einer Mustersprache sind die einzelnen Muster untereinander verbunden und können in Kombination verwendet werden [Ale77][FBB+14].

2.2 Quantencomputing (QC) Mustersprache

Das QC ist ein aufstrebendes Feld der Informatik, das auf den Prinzipien der Quantenmechanik basiert [NC10]. Im Gegensatz zu herkömmlichen Computern, die auf Bits basieren und in diskreten Zuständen arbeiten, verwenden Quantencomputer sogenannte Qubits, die auf den Gesetzen der Quantenphysik beruhen. Diese Qubits haben die einzigartige Eigenschaft, dass sie sich in Superpositionszuständen befinden können, was bedeutet, dass sie gleichzeitig Null und Eins sein können. Darüber hinaus können Qubits auch miteinander verschränkt werden, was es ermöglicht, Informationen zwischen ihnen zu übertragen und durch Operationen auf einem Qubit auch den Zustand anderer Qubits zu beeinflussen.

Diese Eigenschaften des QCs versprechen eine Reihe von Vorteilen gegenüber herkömmlichen Computern, insbesondere bei der Lösung komplexer Probleme. QC könnte die Lösungsgeschwindigkeit für bestimmte Arten von Problemen erheblich verbessern, indem es parallele Berechnungen auf einer viel größeren Skala ermöglicht. Insbesondere in den Bereichen der Kryptographie [Sho94], Suchen [Gro96], der Simulation von Quantensystemen [Zal98], der Optimierung und des maschinellen Lernens [BWP+17] werden Quantencomputer als vielversprechende Werkzeuge angesehen.

Wegen ihrer bemerkenswerten Fähigkeiten stellt das QC ein sehr interessantes Forschungsgebiet dar. Aktuelle Quantencomputer leiden, wie von John Preskill in [Pre18] beschrieben, unter großer Fehleranfälligkeit. Es existieren zahlreiche Herausforderungen, einschließlich der Notwendigkeit der Quantenkohärenz, des Problems der Quantenfehlerkorrektur und der Schwierigkeiten bei der Erzeugung und Aufrechterhaltung von Qubits. Darüber hinaus erfordert die Arbeit mit Quantencomputern ein völlig neues Denken und einen Ansatz zur Problemlösung, was zur Entwicklung von Quantenalgorithmen und -software führt, die speziell auf diese neue Art der Informationsverarbeitung abgestimmt sind. Quantenalgorithmen nutzen häufig gleiche Komponenten, welche sich als bewährte Lösungen für wieder auftretende Herausforderungen darstellen. In seiner Arbeit [Ley19] entwickelte Leymann die oft genutzte Muster wie das *Entanglement*, die entwickelten Muster bilden eine QC-Mustersprache. Diese Entwicklung ist ein wesentlicher Meilenstein für die Etablierung einer Software-Engineering-Disziplin im Bereich der Quantenalgorithmen [Ley19].

2.3 Lösungsraum, Lösungssprache und konkrete Lösung

Die Konzepte von „Lösungsraum“, „Lösungssprache“ und „konkrete Lösung“ sind eng miteinander verbunden und spielen eine große Rolle beim musterbasierten Problemlösen.

Der „Lösungsraum“ stellt den Gesamtumfang aller potenziellen Lösungen für ein gegebenes Problem dar. Innerhalb dieses multidimensionalen Raums kann jede potenzielle Lösung als spezifische Koordinate repräsentiert werden, wobei die Dimensionen des Raumes die verschiedenen Kriterien oder Parameter darstellen, die zur Charakterisierung und Unterscheidung von Lösungen verwendet werden können [FBBL17].

Um die Navigation innerhalb dieses umfangreichen und komplexen Lösungsraums zu erleichtern und zu strukturieren, wird das Konzept der „Lösungssprache“ verwendet. Eine Lösungssprache ist ein systematischer Ansatz zur Organisation und Darstellung von Lösungen. Sie umfasst eine Sammlung von konzeptuellen Werkzeugen und Methoden, die dazu dienen, die Strukturen und Beziehungen zwischen verschiedenen Lösungen zu erkennen und zu beschreiben [FBBL17].

Die „konkreten Lösungen“ stehen im Zentrum der Lösungssprache. Sie repräsentieren spezifische Implementierungen oder Anwendungen von Musterlösungen in bestimmten Kontexten. Wichtig ist hierbei, dass diese konkreten Lösungen nicht mehr technologieagnostisch sind. Sie bauen auf bestimmten Technologien oder Ansätzen auf und sind somit in ihren Möglichkeiten und ihrer Anwendbarkeit spezifischer und begrenzter als die abstrakteren Muster. Konkrete Lösungen bieten spezifische Beispiele und Leitfäden für die Anwendung von Lösungsstrategien bei spezifischen Problemen oder Situationen. Sie können in einer Lösungsdatenbank gespeichert und mit den entsprechenden Mustern verknüpft werden, um die Anwendung und Anpassung von Lösungen in verschiedenen Kontexten zu unterstützen [FBBL17].

Diese Konzepte bieten zusammen einen umfassenden und strukturierten Ansatz zur Erforschung und Anwendung von Lösungen innerhalb des musterbasierten Problemlösungsprozesses, wobei die konkreten Lösungen eine spezifische und technologieabhängige Perspektive einnehmen.

2.4 Aggregation

Aggregation spielt eine entscheidende Rolle bei der Definition, wie verschiedene Lösungen in einer bestimmten Form kombiniert werden können. In diesem Zusammenhang haben Falkenthal et al. in [FBBL19] eine Lösungsalgebra vorgestellt, die einen mathematischen Rahmen bietet, um konkrete Lösungen – also spezifische Implementierungen von Mustern aus einer Mustersprache – zu analysieren und zu kombinieren. Diese Lösungsalgebra legt fest, wie man verschiedene Lösungen zusammenfügt, um eine komplexere oder umfassendere Lösung zu erstellen. Besonders wichtig sind dabei die Aggregationsoperatoren innerhalb der Lösungsalgebra. Sie dienen als Bindeglied zwischen einzelnen Lösungselementen und definieren, wie diese miteinander verknüpft werden können, um eine aggregierte, umfassendere Lösung zu erzeugen [FBBL19]. Durch das systematische Kombinieren dieser Lösungen lassen sich effizientere und robustere Gesamtlösungen erzielen [FBBL19].

2.5 Pattern Atlas

„PatternPedia“ [FBFL15] ist eine kollaborative Software, die darauf abzielt, bestehende Lösungen zu dokumentieren und abstrahierte Muster zu verwalten. Sie wurde konzipiert, um den Prozess der Musteridentifikation und -erstellung zu unterstützen, an dem häufig mehrere Fachexperten

beteiligt sind. Die Software beinhaltet ein Muster-Metamodell, das in UML spezifiziert ist und die Erstellung von Muster-Dokumenten in Form von Wiki-Artikeln ermöglicht. Darüber hinaus umfasst die Software Lösungs- und Muster-Repositories, die auf diesen Metamodellerweiterungen basieren. Benutzer haben die Möglichkeit, nach Mustern zu suchen, basierend auf ihrer Absicht, ihrem Symbol und ihrer Fragestellung. Sie können auch an beliebigen Stellen in den Textelementen der Musterabschnitte Verweise auf andere Muster einfügen. PatternPedia wurde entwickelt, um die Zusammenarbeit bei der Musteridentifikation zu fördern, insbesondere wenn eine persönliche Interaktion nicht möglich ist. Sie ersetzt jedoch nicht die Notwendigkeit persönlicher Gespräche und interaktiver Workshops zur Musteridentifikation und -erstellung [FBFL15].

Leymann et al. [LB21] betonen in ihren Arbeiten die Bedeutung von Mustersprachen bei der Gestaltung spezifischer Architekturen. In diesem Kontext wird die Vielzahl an Mustern aus verschiedenen Sprachen hervorgehoben, was die Notwendigkeit einer Beschränkung auf relevante Muster unterstreicht. Dies führte zur Entwicklung des "Pattern Atlas"[Pla], einer Plattform, die 2020 aus der früheren PatternPedia hervorging. Der Pattern Atlas dient speziell dazu, den Aufbau komplexer Systeme durch den Einsatz von Mustersprachen zu unterstützen, indem er eine strukturierte Auswahl und Anwendung dieser Muster ermöglicht. Diese Entwicklung reflektiert das wachsende Bewusstsein für die Bedeutung von Designmustern in der Software- und Systemarchitektur.

In dieser Arbeit dient der Pattern Atlas als Grundlage für die Mustersprache und die automatisierte Aggregation von Musterimplementierungen im Bereich des QC.

Der aktuelle Pattern Atlas beinhaltet vier verschiedene Mustersprachen, wobei die Mustersprache für Quantencomputing im Rahmen dieser Arbeit von besonderer Bedeutung ist. Diese spezifische Mustersprache vereint Muster, die in einer Reihe von Schlüsselquellen beschrieben sind. [WBLS21a] präsentiert Muster, die sich darauf konzentrieren, wie Daten von Quantencomputern verarbeitet werden. Eine Erweiterung dieser Arbeit wird in [WBLS21b] vorgenommen, wo zwei Muster für die Datenkodierung vorgestellt werden. [WBLV21] fokussiert sich auf die besten Praktiken für Aufteilungsstrategien, indem es Muster vorstellt, die ein gemeinsames Verständnis von hybriden Algorithmen fördern. In [WBLS22] werden drei gängige Kodierungen als Muster präsentiert, die insbesondere in komplexen Bereichen wie dem Quantencomputing helfen, diese neue Technologie und ihr breites Potenzial für Nutzer mit unterschiedlichem Hintergrund zugänglich zu machen. Besonders erwähnenswert ist die Arbeit in [BBL+22], in der drei neue Muster für die Quantenfehlerbehandlung eingeführt werden. Schließlich beschreibt [GBB+23] fünf Muster, die bewährte Lösungen für die Modularisierung, Integration und Übersetzung von Quantenalgorithmus-Implementierungen darlegen. Weitere Informationen und Details zu diesen Mustern können auch auf dem GitHub-Repository des Pattern Atlas [Cona] und in der Publikation [LB21] gefunden werden.

2.6 Open Quantum Assembly Language (OpenQASM)

Open Quantum Assembly Language 2 (OpenQASM-2), ist eine spezialisierte Programmiersprache für Quantencomputer. Sie ermöglicht es, Quantenzustände und -operationen zu beschreiben und zu manipulieren, indem sie Konzepte wie Quantenregister, Quantengatter und Messungen integriert. Diese Register bilden das Herzstück von Quantenberechnungen, indem sie als Sammlungen von Qubits fungieren. Quantengatter wie Hadamard-, CNOT- und Toffoli-Gatter führen grundlegende Operationen auf diesen Qubits aus, was die Grundlage für komplexe Quantenberechnungen bildet.

Ein weiteres wesentliches Merkmal von OpenQASM ist die Fähigkeit, Messergebnisse von Quantenzuständen zu nutzen, um weitere Quantenoperationen oder klassische Berechnungen zu steuern. Dadurch unterstützt OpenQASM sowohl reine Quanten- als auch hybride Quanten-Klassische Programme, was für moderne Quantenalgorithmen unerlässlich ist. Die Sprache ist zudem plattformunabhängig und erweiterbar, was ihre Anwendung auf einer Vielzahl von Quantencomputern und -simulatoren ermöglicht und die Entwicklung in der Quantentechnologie vorantreibt [CBSG17]. Es gibt aktuell zwei Versionen der Sprache, OpenQASM-2 und OpenQASM-3, die jeweils eigene Features und Verbesserungen bieten. Weitere Informationen und die neuesten Entwicklungen können im OpenQASM GitHub-Repository gefunden werden [Conb].

3 Problemstellung und Anforderungen

In diesem Kapitel wird die Problemstellung der Arbeit vorgestellt sowie die erforderlichen Anforderungen erläutert. Es wird darauf eingegangen, warum die Unterstützung konkreter Implementierungen von QC-Mustern im Hinblick auf Integration und Navigation notwendig ist, um eine umfassende Gesamtlösung zu ermöglichen.Des Weiteren werden die damit verbundenen Herausforderungen erläutert. Anschließend werden die abgeleiteten Anforderungen vorgestellt, die die Kriterien der Arbeit zusammenfassend darlegen.

3.1 Problemstellung

In diesem Abschnitt wird die Problemstellung der vorliegenden Arbeit erläutert, und es wird auf die Notwendigkeit eingegangen, die Integration und Navigation konkreter Implementierungen von QC-Mustern zu unterstützen. Dabei werden auch die Bedeutung der Aggregation einer umfassenden Gesamtlösung hervorgehoben und die damit verbundenen Herausforderungen erläutert.

In der Anwendung von Mustern lassen sich zwei Hauptansätze unterscheiden: der traditionelle Ansatz und der konkreten-Lösungen-basierte Ansatz, wie in [FBBL19] erörtert. Beide Ansätze, dargestellt in Abbildung 3.1, umfassen das Studieren von Mustersprachen als ersten Schritt und die gezielte Auswahl spezifischer Muster zur Problemlösung als zweiten Schritt.

Der wesentliche Unterschied zwischen den beiden Ansätzen liegt in der Art und Weise, wie Lösungen generiert bzw. aggregiert werden. Der traditionelle Ansatz basiert auf der Anwendung konzeptioneller Lösungsprinzipien mittels ausgewählter Muster. Hierbei ist es erforderlich, spezifische Lösungen für jedes dieser Muster zu entwickeln und anschließend in einer umfassenden Lösung zusammenzuführen, wie in Schritt 3.a der Abbildung 3.1 dargestellt. Im Gegensatz dazu ermöglicht der konkreten-Lösungen-basierte Ansatz, eine direktere Projektion des ausgewählten Lösungsweges auf einen spezifischen Lösungspfad. Dieser Pfad setzt sich aus einer Sequenz konkreter Lösungen zusammen, die gemeinsam einen definierten Lösungsraum bilden, wie in Schritt 3.b der Abbildung 3.1 dargestellt. Dadurch wird die Navigation von einem Muster zu dessen spezifischen Implementierungen erleichtert und die Wiederverwendung bereits existierender Implementierungen gefördert [FBBL19].

Im Vergleich zur traditionellen Methode erlaubt der konkreten-Lösungen-basierte Ansatz eine effizientere Aggregation konkreter Lösungen [FBBL19]. Dies wird durch Aggregationsbeschreibungen gesteuert oder sogar mithilfe von Aggregationsoperatoren automatisiert werden. Diese Operatoren bieten Funktionen, die die Zusammenführung einzelner Lösungen zu einer Gesamtlösung ermöglichen. Folglich kann, wenn eine Abfolge konkreter Lösungen, die auf einer zuvor ausgewählten Mustersprache basiert, identifiziert wird, durch den Einsatz von Aggregationsoperatoren eine aggregierte konkrete Lösung für den definierten Lösungspfad generiert werden.

Die in „Pattern Atlas“ [Pla] beschriebene QC-Mustersprache ermöglicht das Verstehen bzw. das Studieren von QC-Mustern, was dem ersten Schritt der Musteranwendung entspricht, wie in Abbildung 3.1 dargestellt. Jedoch wird hier eine entscheidende Einschränkung deutlich: Es fehlt die Möglichkeit, Muster auszuwählen und zu kombinieren, um eine umfassende Lösung zu entwickeln. Während eine traditionelle Musteranwendung mit Hilfe des Pattern Atlas theoretisch möglich ist, bleibt die Implementierung konkreter Lösungen auf Basis der Muster unzugänglich.

Diese Einschränkung ist besonders für Anwender, die nicht mit den Feinheiten des Quantencomputings vertraut sind, problematisch [Ley19]. Die Herausforderung liegt darin, QC-Algorithmen eigenständig zu implementieren [Ley19]. Dies hebt hervor, dass die Verwendung der QC-Mustersprache aus dem Pattern Atlas zwar ein hilfreiches Werkzeug für das Verständnis ist, jedoch zwingt sie den Anwender, alle Implementierungen selbstständig durchzuführen. Es besteht also ein klarer Bedarf an einer erweiterten Methodik oder einem Toolset, das es auch weniger erfahrenen Anwendern ermöglicht, effektiv QC-Muster zu kombinieren und zu nutzen, um funktionierende Quantenalgorithmen zu entwickeln.

3.2 Anforderungen

In diesem Abschnitt werden die Anforderungen an die vorliegende Arbeit definiert. Hierbei werden sowohl Anforderungen an das Gesamtkonzept als auch an spezifische Aspekte erläutert.

Einen Überblick über diese Anforderungen gibt Tabelle 3.1, und im Folgenden werden sie ausführlicher diskutieren:

#	Anforderung
A1	Auswahl von Mustern bzw. konkreten Lösungen soll mithilfe einer UI ermöglicht werden.
A2	Die konkreten Lösungen der ausgewählten Muster sollen zu einer Gesamtlösung aggregiert werden.
A3	Die aggregierte Gesamtlösung soll lauffähig sein und den Architekturentwurf implementieren.

Tabelle 3.1: Ein Überblick über die Anforderungen auf der konzeptionelle Ebene.

A1 Auswahl von Mustern bzw. konkreten Lösungen soll mithilfe einer grafischen UI ermöglicht werden.

Um die Benutzerfreundlichkeit zu erhöhen, sollte die Plattform eine grafische UI bereitstellen, über die der Nutzer die Auswahl der Muster treffen kann. Die Interaktion mit grafischen Elementen erleichtert die Auswahl und Zuordnung der gewünschten Muster und die dazugehörigen konkreten Lösungen und erhöht die Übersichtlichkeit des Entwurfs.

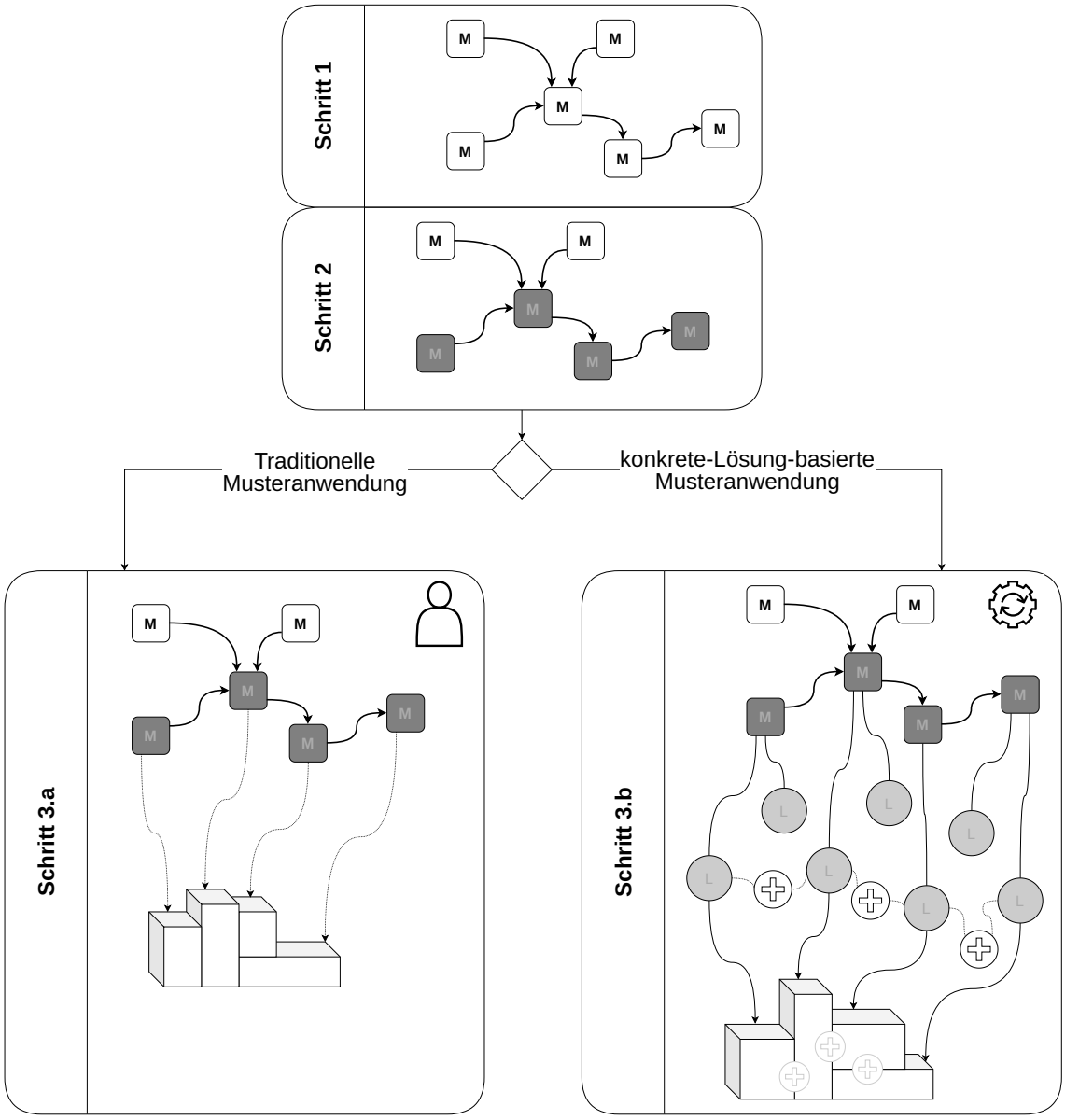

Abbildung 3.1: Vergleich der traditionellen und konkreten-Lösungen-basierten Ansätze zur Musteranwendung: Beide beginnen mit dem Studium von Mustersprachen (Schritt 1) und der Auswahl spezifischer Muster (Schritt 2) für die Problemlösung. Der traditionelle Ansatz entwickelt spezifische Lösungen für jedes Muster, die zu einer Gesamtlösung zusammengeführt werden (Schritt 3.a), während der konkreten-Lösungen-basierte Ansatz auf der direkten Anwendung einer Sequenz konkreter Lösungen basiert, die einen definierten Lösungsraum bilden (Schritt 3.b), und ermöglicht eine effizientere Aggregation dieser Lösungen durch Aggregationsbeschreibungen und -operatoren. Adaptiert von [FBBL19]

A2 Die konkreten Lösungen der ausgewählten Muster sollen zu einer Gesamtlösung aggregiert werden.

Die Plattform sollte Mechanismen bieten, um die ausgewählten konkreten Lösungen zu einer Gesamtlösung zu aggregieren. Dies bedeutet, dass die einzelnen Lösungen der Muster in einer sinnvollen Weise miteinander verknüpft werden, um den gewünschten Architekturentwurf zu erstellen.

A3 Die aggregierte Gesamtlösung soll lauffähig sein und den Architekturentwurf implementieren.

Es ist von entscheidender Bedeutung, dass die aggregierte Gesamtlösung, die aus den ausgewählten konkreten Lösungen besteht, auch tatsächlich lauffähig ist und den gewünschten Architekturentwurf vollständig implementiert. Die Plattform sollte Mechanismen zur Überprüfung der Funktionalität bieten und sicherstellen, dass die erstellte Lösung den spezifizierten Anforderungen entspricht.

4 Konzept

Zur Realisierung des Frameworks wurden mehrere bestehende Komponenten erweitert, andere lediglich verwendet und eine neue Komponente hinzugefügt. Die Zusammensetzung und das Zusammenspiel dieser Komponenten sind in Abbildung Abbildung 4.1 dargestellt.

Die Open-Source-Anwendung Pattern Atlas [Pla] setzt sich in vereinfachter Darstellung aus drei Hauptkomponenten zusammen: dem Pattern Atlas UI, dem Pattern Atlas Application Programming Interfaces (API) und einer relationalen Datenbank zur Sicherstellung der Datenpersistenz. Ähnlich ist auch der QC Atlas [QuA] strukturiert. Die Benutzerschnittstellen beider, des Pattern Atlas und des QC Atlas, sind als Webanwendungen mit dem Framework Angular [Tea] entwickelt worden, was eine interaktive Nutzererfahrung ermöglicht. Die Komponenten Pattern Atlas API und QC Atlas API fungieren als REST-Schnittstellen (Representational State Transfer) und sind essenziell für die Verwaltung vielfältiger Daten, darunter Mustersprachen, Muster, Beziehungen und konkrete Lösungen. Beide APIs sind Java-Anwendungen, entwickelt mit dem Framework Spring Boot [Piv]. Zur Datenspeicherung wird eine relationale Datenbank eingesetzt, die mit Structured Query Language (SQL) betrieben wird, wobei standardmäßig PostgreSQL [Pos] zum Einsatz kommt.

Die Open-Source-Anwendung Winery [Con23] bietet eine webbasierte Plattform für die graphbasierte Modellierung von Topologien und die Definition wiederverwendbarer Komponenten- und Beziehungstypen [KBBL13]. In dieser Arbeit wird Winery zur Modellierung von QC-Mustern und ihren Beziehungen eingesetzt. Das speziell für dieses Projekt entwickelte Repository [Aldb], gehostet auf GitHub [Incb], fungiert als Speicherort für die Definitionen der Winery-QC-Knoten und -Beziehungstypen. Diese Definitionen werden beim Start von Winery geladen und sind dann innerhalb der Anwendung zur Verwendung verfügbar.

Die Komponente Aggregation Service [Ahm23] ist eine neue API, die in dieser Arbeit entwickelt wurde. Sie dient zur Aggregation von konkreten Lösungen, die in Form von OpenQASM-2 [CBSG17] Abschnitt 2.6 vorliegen. Bei der Aggregation kann die Aggregation die benötigten Dateien mittels Hypertext Transfer Protocol (HTTP) aus dem QC-Atlas API abrufen.

Ein zentrales Element in der Architektur dieses Frameworks [Alda] ist die Verwendung von Docker [Inca], welches die konsistente und plattformunabhängige Ausführung aller Komponenten durch Containerisierung ermöglicht. Für die Kommunikation zwischen der API und der Datenbank wird das Spring Data JPA (Java Persistence API) (JPA) eingesetzt, das eine vereinfachte Datenmanipulation und -abfrage ermöglicht. Darüber hinaus interagieren die Komponenten mittels HTTP, was eine flexible und standardisierte Kommunikation zwischen den verschiedenen Teilen des Systems sicherstellt.

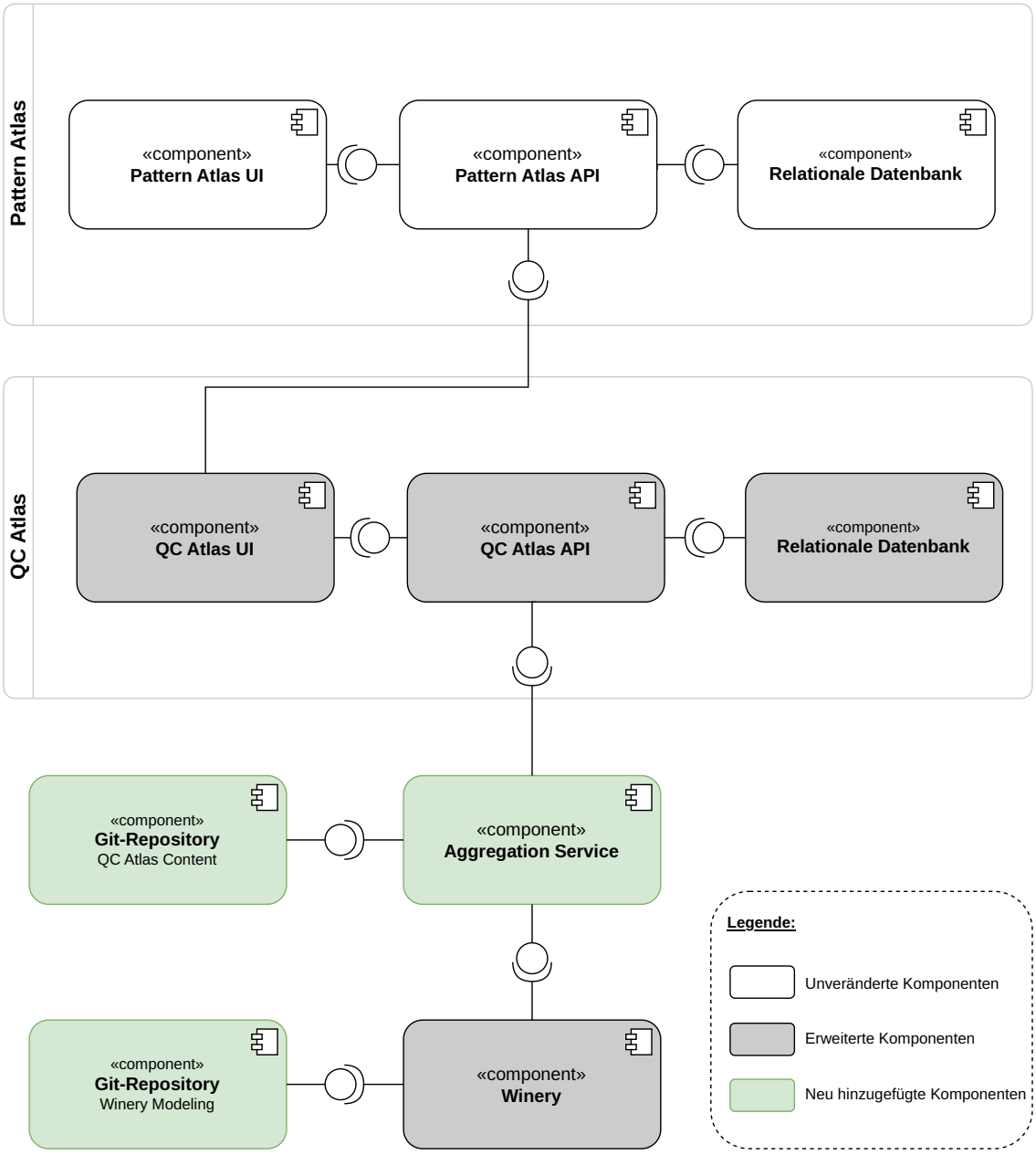

Abbildung 4.1: Das Komponentenmodell des Frameworks setzt sich aus mehreren Elementen zusammen: dem Pattern bzw. QC Atlas UI, der Pattern bzw. QC Atlas API, und jeweils einer relationalen Datenbank zur Datenpersistenz. Zusätzlich werden die Winery-Komponente und ein GitHub-Repository für die Modellierung der Knoten und deren Beziehungen genutzt. Die Aggregation-Komponente wird neu implementiert.

5 Implementierung

In diesem Kapitel erfolgt die Beschreibung der Implementierung der in Kapitel 4 vorgestellten Konzepte. Zunächst wird die Erweiterung des QC Atlas vorgestellt. Anschließend wird die Erweiterung der Winery beschrieben. Abschließend wird die Implementierung des Aggregation Service erläutert.

5.1 QC Atlas Erweiterung

Als Ausgangspunkt für die Implementierung wurde der QC Atlas [QuA] gewählt. Die Komponenten des QC Atlas, die in Kapitel 4 vorgestellt wurden, sind nun so erweitert worden, dass es möglich ist, konkrete Lösungen zu hinterlegen. Ein Benutzer kann über einen Browser auf die QC Atlas UI zugreifen und zu dem Pattern-Button navigieren. Dort werden alle aktuellen QC-Muster des Pattern Atlas [Pla] angezeigt. Der Benutzer hat die Möglichkeit, ein Muster auszuwählen und darauf zu klicken, um beliebig viele konkrete Lösungen für dieses Muster hinzuzufügen. Diese konkreten Lösungen werden somit einem spezifischen Muster zugeordnet. Nachdem eine konkrete Lösung hinterlegt wurde, besteht die Möglichkeit, eine (OpenQASM-2)-Datei auszuwählen und dieser Lösung anzuhängen. Ein Sequenzdiagramm, das diesen Prozess darstellt, ist in Abbildung 5.1 gezeigt. Ergänzend dazu illustrieren mehrere Screenshots der QC Atlas UI [QuA] verschiedene Aspekte der Erweiterung: Abbildung 5.2 zeigt einen Screenshot, auf dem die erweiterte Anzeige der Muster und deren aktuelle QC-Muster im Pattern Atlas ersichtlich ist. In Abbildung 5.3 wird ein Screenshot präsentiert, der die Darstellung konkreter Lösungen für ein ausgewähltes Muster aufzeigt. Der Dialog zum Hinzufügen einer konkreten Lösung für ein Muster ist in Abbildung 5.4 durch einen Screenshot dargestellt. Abbildung 5.5 veranschaulicht mittels eines Screenshots den Dialog zum Anhängen einer Datei an eine konkrete Lösung. Abschließend zeigt Abbildung 5.6 einen Screenshot, der die Übersicht der zu einer konkreten Lösung gehörenden Dateien abbildet.

Das Git Repository QC Atlas Content [Aldc], gehostet auf GitHub [Incb], fungiert als Backup-Speicherort für das Datenbankschema und die Daten des QC Atlas. Es dient auch als Speicherort für die (OpenQASM-2)-Dateien der konkreten Lösungen.

5.2 Winery Erweiterung

Im Winery Modeling Repository [Aldb] wurden spezifische Knoten definiert, die den aktuellen QC-Mustern des Pattern Atlas [Pla] entsprechen. Ein Beispiel für eine solche Knoten-Definition ist in Listing 5.1 dargestellt. In diesem XML-Ausschnitt ist das Element <Definitions> der Ausgangspunkt, das verschiedene Namensräume beinhaltet, beispielsweise der targetNamespace ist ein wichtiger Bestandteil der XML-Definition. Im gegebenen Beispiel ist der targetNamespace

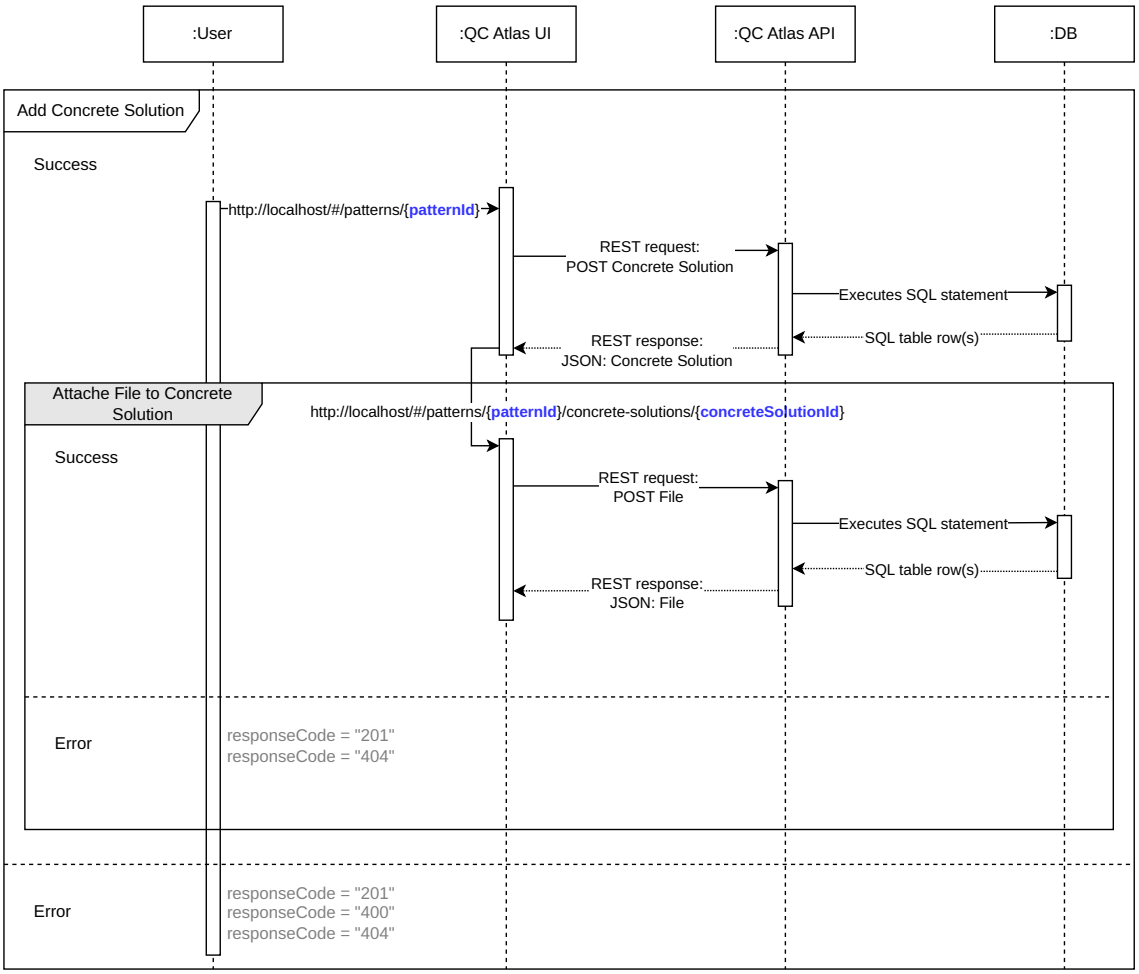

Abbildung 5.1: Sequenzdiagram für Hinterlegung eines Konkreten Lösung und das Anhängen von Datei.

mit `https://bloqcat.github.io/tosca/nodetypes` angegeben. Im Kontext von TOSCA spezifiziert der targetNamespace den Namensraum, in dem die NodeTypes definiert sind. Dadurch wird sichergestellt, dass die Namen der NodeTypes einzigartig sind und innerhalb des definierten Namensraums interpretiert werden.

Zusätzlich wurden zwei relevante Beziehungstypen für diese Arbeit definiert: *Aggregation* und *Concrete Solution of*. Ein Beispiel für diese Beziehungstypen findet sich in Listing 5.2. Im gezeigten XML-Beispiel wird der Beziehungstyp Aggregation definiert. Diese Definition legt die Eigenschaften der Beziehung fest, wie ihre Benennung, Darstellungsattribute (wie Pfeilspitzen und Farben) und ob sie als abstrakt oder endgültig angesehen wird. Die Verwendung des targetNamespace gewährleistet, dass dieser Beziehungstyp klar und eindeutig innerhalb des TOSCA-Modells identifiziert wird.

Diese Definitionen werden beim Start von Winery automatisch geladen und stehen anschließend in der Anwendung zur Verfügung. Die Abbildungen Abbildung 5.7 und Abbildung 5.8 illustrieren sowohl die Knoten als auch die Relationen in der Winery UI [Con23], wie sie in [Aldb] definiert sind.

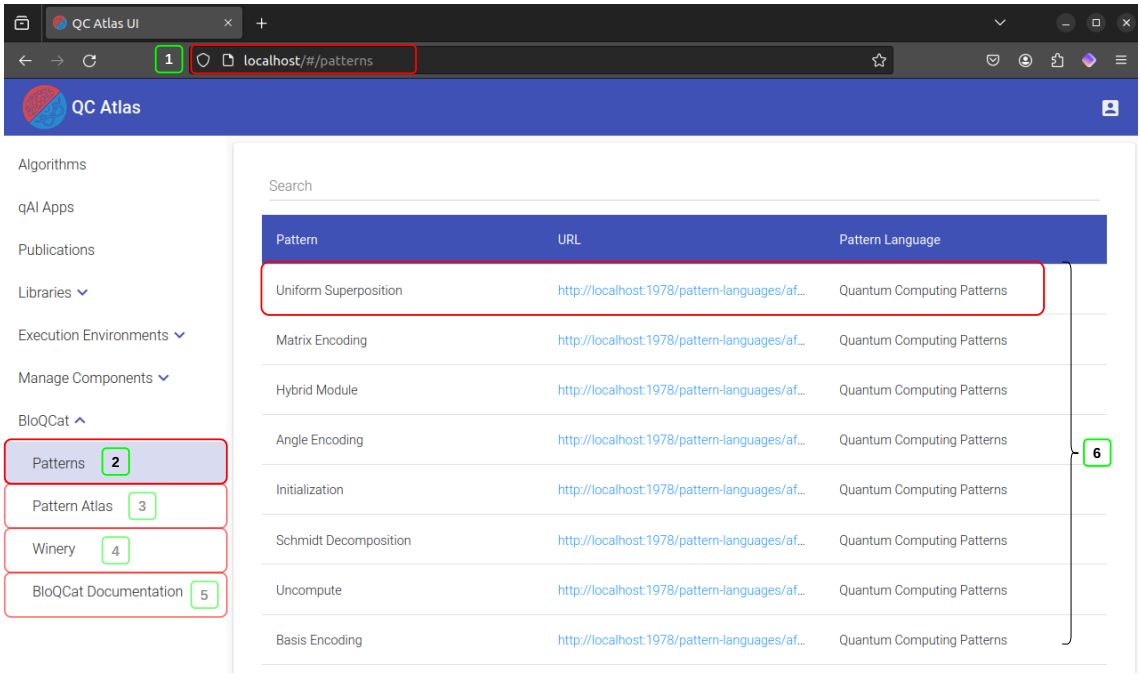

Abbildung 5.2: Screenshot von QC Atlas UI [QuA]. (1) Die URL, um auf die UI zuzugreifen. (2) Navigation zu den verfügbaren QC-Mustern aus dem Pattern Atlas [Pla]. (3) Verknüpfung zum Pattern Atlas [Pla], welche in einem neuen Browser-Tab geöffnet wird. (4) Verknüpfung zu Winery [KBBL13], welche in einem neuen Browser-Tab geöffnet wird. (5) Verknüpfung zur Dokumentation des Frameworks, welche in einem neuen Browser-Tab geöffnet wird. (6) Anzeige einer Teilmenge der QC-Muster.

Listing 5.1 Winery Knoten Definition Beispiel.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Definitions
    xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
    xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"
    xmlns:selfservice="http://www.eclipse.org/winery/model/selfservice"
    xmlns:testwineryopentoscaorg="http://test.winery.opentosca.org"
    targetNamespace="https://bloqcat.github.io/tosca/nodetypes"
    id="ntyIgeneral1-UniformSuperposition_latest-w1-wip1">
    <NodeType name="UniformSuperposition_latest-w1-wip1"
    abstract="no" final="no"
    targetNamespace="https://bloqcat.github.io/tosca/nodetypes"/>
</Definitions>
```

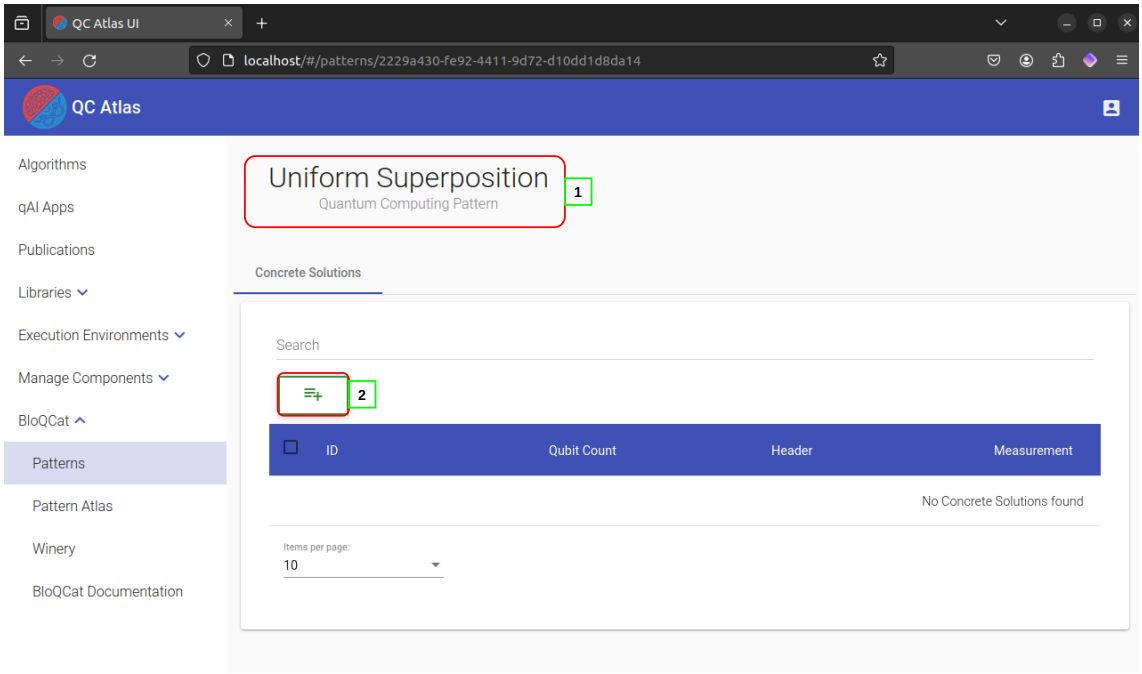

Abbildung 5.3: Screenshot von QC Atlas UI [QuA]. (1) Das geklickte Muster Name. (2) Ein Button um konkrete Lösungen für das Muster hinzuzufügen.

Listing 5.2 Winery Beziehungs Definition Beispiel.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Definitions
    xmlns="http://docs.oasis-open.org/tosca/ns/2011/12"
    xmlns:winery="http://www.opentosca.org/winery/extensions/tosca/2013/02/12"
    xmlns:selfservice="http://www.eclipse.org/winery/model/selfservice"
    xmlns:testwineryopentoscaorg="http://test.winery.opentosca.org"
    targetNamespace="https://bloqcat.github.io/tosca/relationshiptypes"
    id="ntyIgeneral1-Aggregation_latest-w1-wip1">

    <RelationshipType
    name="Aggregation"
    abstract="no"
    final="no"
    targetNamespace="https://bloqcat.github.io/tosca/relationshiptypes"
    winery:targetArrowHead="none"
    winery:sourceArrowHead="none"
    winery:color="#000000"
    winery:hoverColor="black"
    winery:dash="dotted"/>
</Definitions>
```

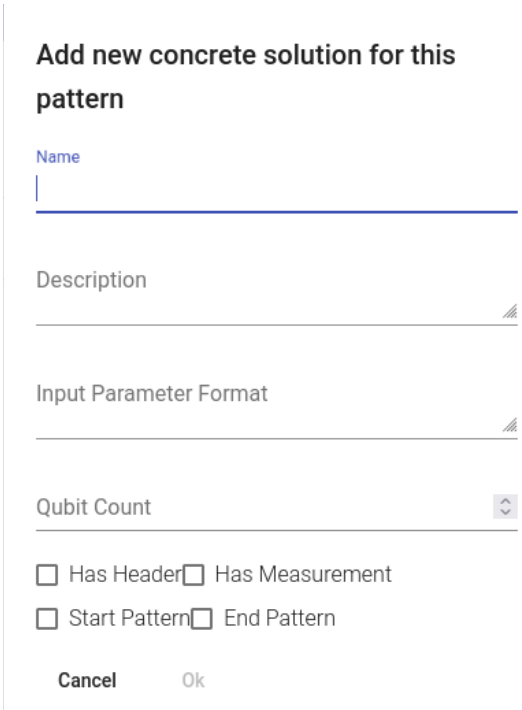

Abbildung 5.4: Screenshot von QC Atlas UI [QuA]. In diesem Dialog kann der Benutzer eine konkrete Lösung für das Muster hinzufügen, wobei alle Felder außer dem Qubit Count optional sind sind.

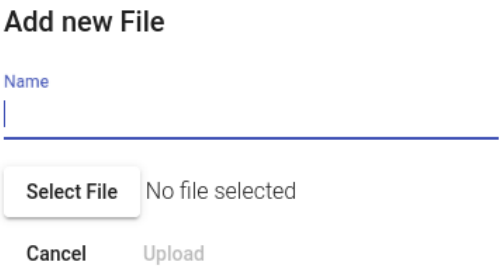

Abbildung 5.5: Screenshot von QC Atlas UI [QuA]. In diesem Dialog kann der Benutzer eine Datei für die konkrete Lösung hinzufügen.

Der Winery Topologiemodeller bietet die Möglichkeit, Knoten und Beziehungen zu einem Topologiemodell zusammenzustellen. Dank der vordefinierten Elemente aus dem Winery Modeling Repository stehen spezielle Knoten und Beziehungstypen für QC-Muster zur Verfügung, die im Topologiemodell zusammengestellt und miteinander verbunden werden können. Dieser Prozess führt zur Entwicklung einer Mustersprache, die in Abschnitt 2.1 detailliert beschrieben wird. Diese Mustersprache erlaubt das Beschreiben von QC-Algorithmen. Basierend auf einer solchen Mustersprache kann eine Lösungssprache generiert werden. Die Generierung der Lösungssprache erfolgt in mehreren Schritten. Zunächst werden die Knoten aus dem Topologiemodell extrahiert. Anschließend werden die IDs dieser Knoten aus dem Pattern Atlas [Pla] abgefragt. Diese IDs sind notwendig, um die hinterlegten konkreten Lösungen für das jeweilige Muster aus dem QC Atlas [QuA] abzurufen. Nachdem die konkreten Lösungen für die Knoten aus dem QC Atlas erhalten wurden, wird für jede konkrete Lösung ein Knotenobjekt erstellt, das die Metadaten der Lösung enthält. Zudem wird eine Beziehung „Concrete Solution of“ generiert, welche die konkrete Lösung mit dem jeweiligen Muster verbindet. Diese neu erstellten Elemente werden anschließend in das Topologiemodell integriert und im Topologiemodeller visualisiert. Der Pseudocode für die Generierung der Lösungssprache ist in Algorithmus 5.1 dargestellt.

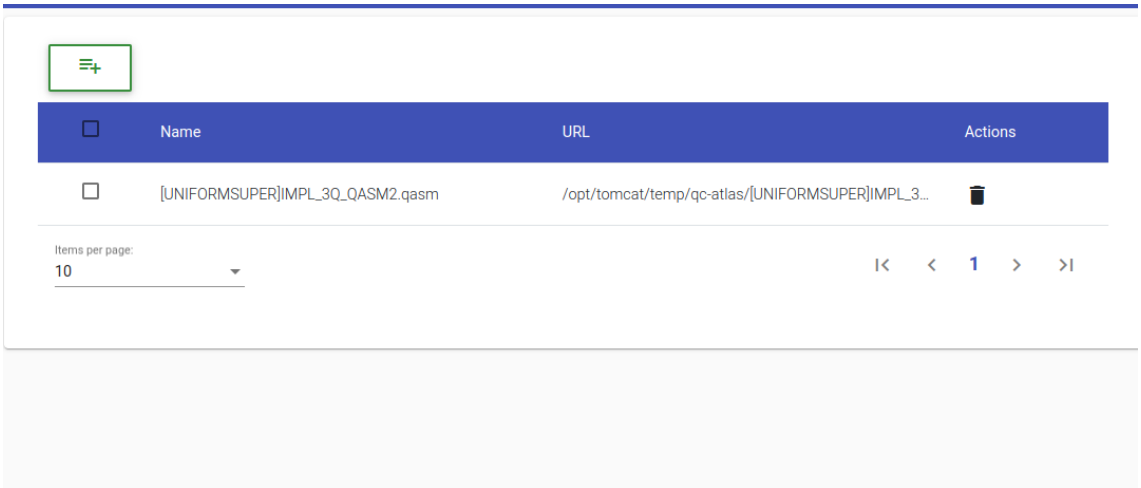

Abbildung 5.6: Screenshot von QC Atlas UI [QuA]. Zeigt die Dateie, die zu einer bestimmten konkreten Lösung gehört.

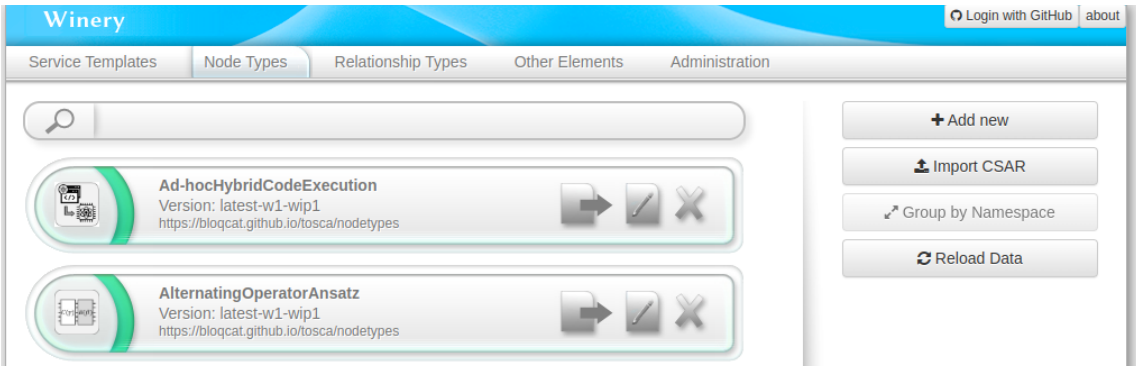

Abbildung 5.7: Screenshot von Winery [Con23] UI, auf dem die in [Aldb] definierten Knoten dargestellt sind.

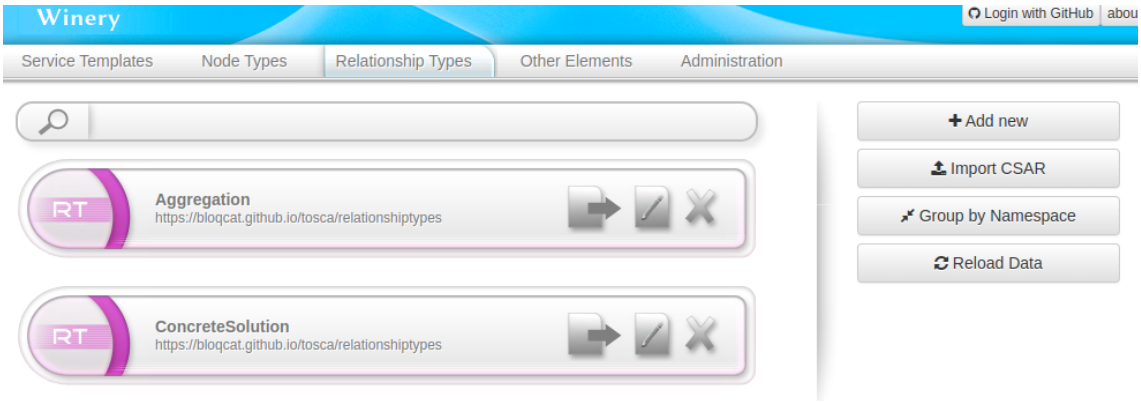

Abbildung 5.8: Screenshot dvon Winery [Con23] UI, auf dem die in [Aldb] definierten Beziehungen dargestellt sind.

Algorithmus 5.1 Pseudocode für die Generierung von Lösungsprache.

Eingabe: *Ein Winery-Topologiemodell-Objekt, das mehrere Knoten bzw. Muster enthält.*
Ausgabe: *Das modifizierte Topologiemodell, in dem für alle vorhandenen Muster die verfügbaren konkreten Lösungen hinzugefügt werden. Zusätzlich werden diese konkreten Lösungen mit den entsprechenden Mustern durch eine Beziehung namens „Concrete Solution of" verbunden.*

procedure GENERATESOLUTIONLANGUAGE(T)
  $\mathcal{V} \leftarrow \{t \in T \mid t \text{ is a node}\}$
  for all $v \in \mathcal{V}$ **do**
    $id(k) \leftarrow$ PATTERNATLASAPI($getIdOf(k)$)
  end for
  for all $v \in \mathcal{V}$ **do**
    $CSs(v) \leftarrow$ QCATLASAPI($getConcreteSolutionsOf(k)$)
    $T \leftarrow CSs(v)$
    for all $cs \in CSs(v)$ **do**
      $R(v) \leftarrow$ CREATERELATIONSHIP(v, cs)
    end for
    $T \leftarrow R(v)$
  end for
end procedure

Nachdem eine Lösungsprache generiert wurde, kann der Benutzer konkrete Lösungen für die jeweiligen Muster auswählen und miteinander verknüpfen. Dabei ist es wichtig, dass die ausgewählten konkreten Lösungen kompatibel sind, beispielsweise in Bezug auf die Anzahl der Qubits. Nicht relevante konkrete Lösungen können aus dem Topologiemodell entfernt werden. Nach Abschluss des Entwurfs des QC-Algorithms kann der Benutzer diesen an den Aggregation Service zur Aggregation senden (siehe Abschnitt 5.3). Der Benutzer erhält daraufhin entweder eine aggregierte Gesamtlösung oder eine Fehlermeldung, die beispielsweise auf eine Inkompatibilität der ausgewählten konkreten Lösungen hinweisen könnte.

5.3 Aggregation Service

Die Aggregation Service Komponente [Ahm23] ist eine im Rahmen dieser Arbeit neu entwickelte API. Ihr Hauptzweck liegt in der Aggregation von konkreten Lösungen, die in der Form von OpenQASM-2 [CBSG17] vorliegen, wie in Abschnitt 2.6 dargestellt. Die Implementierung der Aggregation-Komponente erfolgte in Python und nutzt das Flask-Framework [Pal]. Ein wesentlicher Bestandteil der API ist der POST-Endpoint `/bloqcat/winery/topology/deploy/json`. Dieser Endpoint ermöglicht es Nutzern, ein Topologiemodell im JSON-Format zu übermitteln. Das Modell sollte sowohl die Mustersprache als auch die Lösungssprache beinhalten. Die Aggregation wird im Endpoint in mehreren Schritten durchgeführt. Zunächst erfolgt eine gründliche Überprüfung der Eingabedaten auf Korrektheit. Dabei wird unter anderem kontrolliert, ob das Topologiemodell bereits eine Lösungssprache, d.h. konkrete Lösungen, die mit der Beziehung „Aggregation" verbunden sind, enthält und ob die Anzahl der Qubits übereinstimmt. Anschließend werden die Knoten und Beziehungen aus dem Topologiemodell extrahiert. Danach wird ein sogenannter Lösungspfad aus dem Topologiemodell abgeleitet, der alle konkreten Lösungen, die mit der Beziehung

Algorithmus 5.2 Pseudocode für die Aggregation.

Eingabe: *Ein Winery-Topologiemodell-Objekt, das mehrere Knoten bzw. Muster enthält. Diese konkreten Lösungen sind mit der Beziehung „Aggregation“ verbunden.*
Ausgabe: *Eine aggregierte Gesamtlösung, die aus den verbundenen konkreten Lösungen besteht, dargestellt in Form von OpenQASM-2 [CBSG17].*

procedure AGGREGATION(T)
  VALIDATEDATA(T) // validates a dataset to ensure it meets specific criteria
  if T is valid **then**
    $\mathcal{V} \leftarrow \{t \in T \mid t \text{ is a concrete solution node}\}$
    $\mathcal{E} \leftarrow \{t \in T \mid t \text{ is a aggregation relationship}\}$
  end if
  $P \leftarrow$ EXTRACTSOLUTIONPATH($\mathcal{V}, \mathcal{E}$) // Extracts the solution path from the topology
  for all $p \in P$ **do**
    $CSsFiles \leftarrow$ QCATLASAPI(p)
  end for
  aggregationResult $\leftarrow header + qreg + creg + cs_1 + cs_2 + \cdots + cs_n + measurement$
end procedure

„Aggregation“ verbunden sind, umfasst. Die zugehörigen Dateien des Lösungspfades werden aus dem QC Atlas [QuA] abgefragt. Für die Ausgabe wird zunächst ein „Header“ konstruiert, gefolgt von der Definition des Quantum Registers und des klassischen Registers. Anschließend werden die konkreten Lösungen in der Reihenfolge, wie sie im Lösungspfad vorkommen, in die Ausgabe eingefügt. Abschließend wird ein Measurement hinzugefügt, bei dem alle Qubits gemessen werden. Der Prozess der Aggregation wird in Algorithmus 5.2 detailliert beschrieben. Zusätzlich werden in der Ausgabe Kommentare eingefügt, die den Ursprung jeder Komponente erläutern. Dies erleichtert das Nachvollziehen des Aggregationsprozesses und bietet eine hilfreiche Grundlage für eventuelles Debugging.

6 Anwendungsszenario

In diesem Kapitel wird detailliert beschrieben, wie Benutzer das Framework nutzen können, um ein QC-Algorithmus zu implementieren. Dies umfasst die Initialisierung der Framework-Komponenten, die Erstellung eines geeigneten Topologiemodells im Winery UI [Con23] und die Auswahl der erforderlichen Muster und konkreten Lösungen. Besonders hervorzuheben ist die Fähigkeit unseres Frameworks, eine Lösungssprache zu generieren und anschließend einer Gesamtlösung zu aggregieren, die an einen Quantencomputer gesendet werden kann, um den QC-Algorithmus auszuführen.

6.1 Der Deutsch-Algorithmus als Anwendungsbeispiel

Der Deutsch-Algorithmus, einer der ersten Algorithmen, der speziell für Quantencomputer entwickelt wurde, demonstriert eindrucksvoll das Überlegenheitspotenzial von Quantencomputern gegenüber klassischen Computern. Dieser Algorithmus adressiert das Problem, zu bestimmen, ob eine gegebene Funktion $f : \{0, 1\} \rightarrow \{0, 1\}$ konstant oder balanciert ist. Eine konstante Funktion gibt für alle Eingaben denselben Wert aus, während eine balancierte Funktion für die Hälfte der Eingaben 0 und für die andere Hälfte 1 zurückgibt [Hom08].

Nun beabsichtigen wir, den Deutsch-Algorithmus oder eine seiner Varianten - wie etwa die in [Hom08] beschriebene - mithilfe unseres entwickelten Frameworks zu implementieren. Bevor wir dieses Szenario jedoch detaillierter ausführen, ist es wichtig zu erwähnen, dass im QC Atlas [QuA] bereits geeignete konkrete Lösungen hinterlegt sein sollten, insbesondere für den Deutsch-Algorithmus. Konkret sind passende Lösungen für die Muster *Uniform Superposition* und *Oracle* erforderlich.

Zu Beginn werden die Komponenten des Frameworks [Alda] initialisiert. Für die Konstruktion des Deutsch-Algorithmus unter Verwendung von Mustern ist zunächst die Erstellung eines geeigneten Topologiemodells erforderlich. Der Benutzer kann hierfür zur Winery navigieren, beispielsweise durch Anklicken des „Winery“-Buttons in der QC Atlas UI, wie in Abbildung 5.2 illustriert. Dies öffnet die Winery UI in einem neuen Browser-Tab, dargestellt in Abbildung 6.1. In der Winery UI hat der Benutzer die Möglichkeit, ein neues „Service Template“ zu erstellen, wobei die Eingabe eines Namens notwendig ist, wie in Abbildung 6.2 gezeigt. Nach der Erstellung des „Service Template“ präsentiert sich die UI entsprechend Abbildung 6.3.

Für die Bearbeitung das Topologiemodell navigiert der Benutzer zum „Topology Template“ und wählt „Open Editor“. Der Winery Topologiemodeller öffnet sich daraufhin, wie in Abbildung 6.4 abgebildet. Hier kann der Benutzer die erforderlichen Knoten bzw. Muster für das Topologiemodell auswählen, beispielsweise die Knoten *Uniform Superposition* und *Oracle* für den Deutsch-Algorithmus, dargestellt in Abbildung 6.5. Diese Muster können anschließend miteinander verbunden werden, wie in Abbildung 6.6 illustriert. Nach Abschluss der Topologiemodellerstellung kann der Benutzer

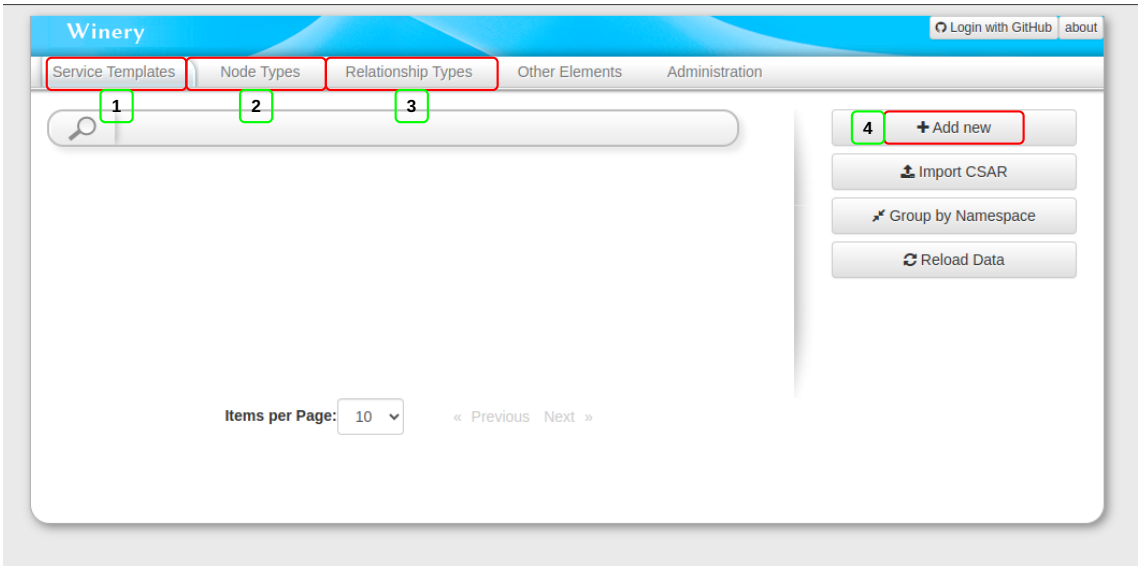

Abbildung 6.1: Screenshot der UI von Winery [Con23]. (1) Anzeige der verfügbaren „Service Templates“ bzw. Topologiemodellen. (2) Darstellung der verfügbaren Knoten, die gemäß [Aldb] definiert sind. (3) Darstellung der verfügbaren Relationstypen, ebenfalls definiert in [Aldb]. (4) Ein Button zur Erstellung einer neuen „Service Templates“ bzw. Topologiemodell.

die Lösungssprache generieren, indem er auf „Generate Solution Language“ klickt (siehe Button Nr. 2 in Abbildung 6.4). Dies ist möglich, da gemäß unserer Annahme im QC Atlas [QuA] bereits konkrete Lösungen für die Muster *Uniform Superposition* und *Oracle* existieren. Die Generierung der Lösungssprache wird in Algorithmus 5.1 demonstriert.

Die erzeugte Lösungssprache wird im Winery Topologiemodeller angezeigt, wie in Abbildung 6.7 und Abbildung 6.8 zu sehen ist. In Abbildung 6.8 wird außerdem gezeigt, dass die Winery Topologiemodeller UI die Einsicht in Metadaten der konkreten Lösungen, wie beispielsweise die Anzahl der Qubits, ermöglicht. Diese Metadaten sind bei der Auswahl der konkreten Lösungen für die Aggregation einer Gesamtlösung entscheidend. Sobald die benötigten konkreten Lösungen ausgesucht wurden, müssen sie mittels der „Aggregation“-Beziehung verbunden werden, wie in Abbildung 6.9 illustriert. Konkrete Lösungen, die nicht verbunden sind, gelten als nicht ausgewählt und werden bei der Aggregation zur Gesamtlösung nicht berücksichtigt. Nachdem die Verbindungen hergestellt sind, kann der Benutzer den „Deploy“-Button betätigen. Dieser Schritt aggregiert die ausgewählten konkreten Lösungen zu einer Gesamtlösung, die als Antwort auf den „Deploy“-Request im Browser angezeigt wird, wie in Abbildung 6.10 illustriert. Diese aggregierte Gesamtlösung kann dann an einen Quantencomputer übermittelt werden, um den Deutsch-Algorithmus auszuführen.

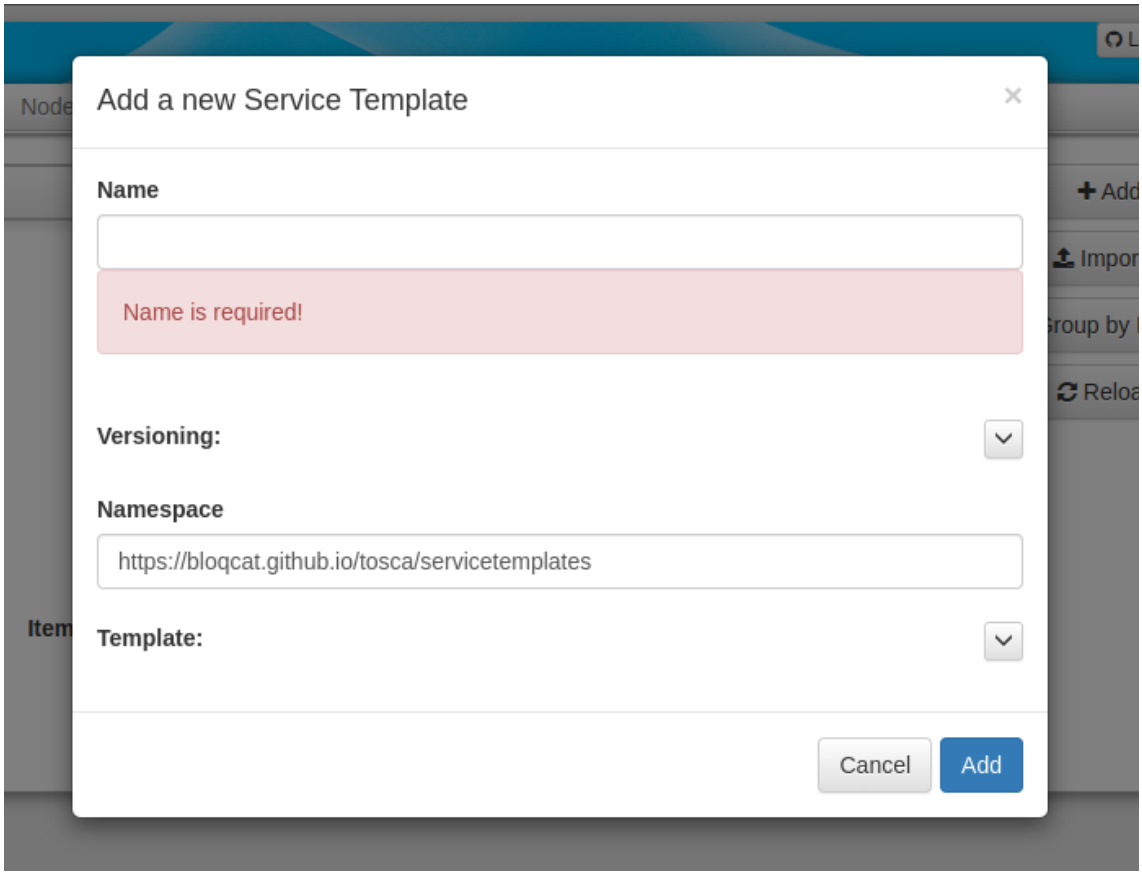

Abbildung 6.2: Screenshot der UI von Winery [Con23], aufgenommen in dem Moment, als ein Benutzer auf den Button „+ Add new“ klickt, um ein neues „Service Template“ bzw. ein neues Topologiemodell zu erstellen.

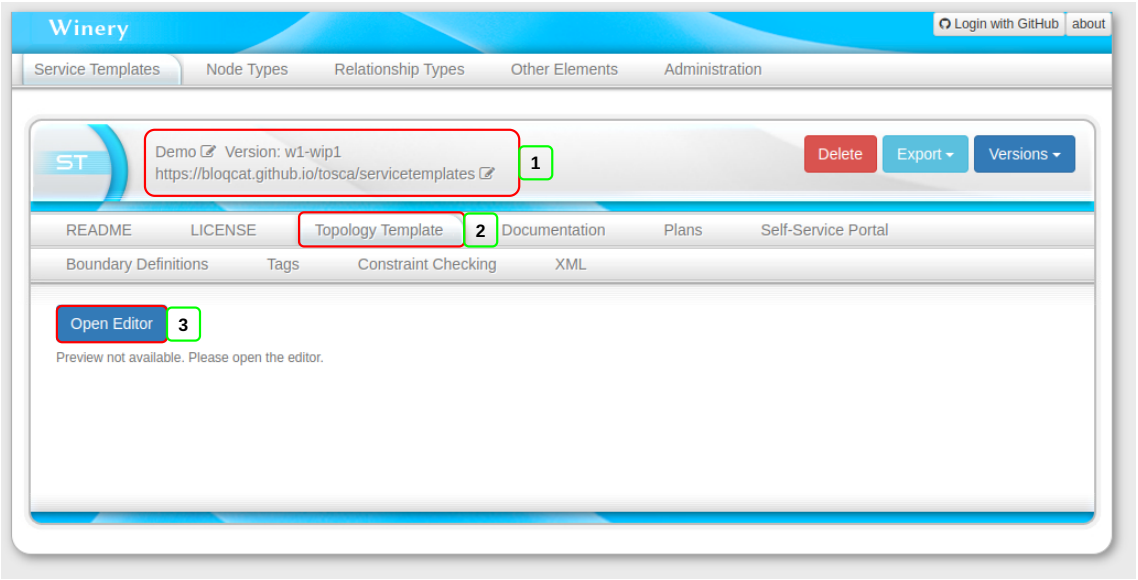

Abbildung 6.3: Screenshot der UI von Winery [Con23]. aufgenommen in dem Moment, nachdem ein Benutzer ein neues „Service Template“ mit dem Namen „Demo“ erstellt hat. (1) Name, Version und Namespace der „Service Template“ (2) Topology Template Rubrik. (3) Öffnet der Winery Topologiemodeller um eine Topologiemodell für das „Service Template“ zu erstellen oder zu ändern.

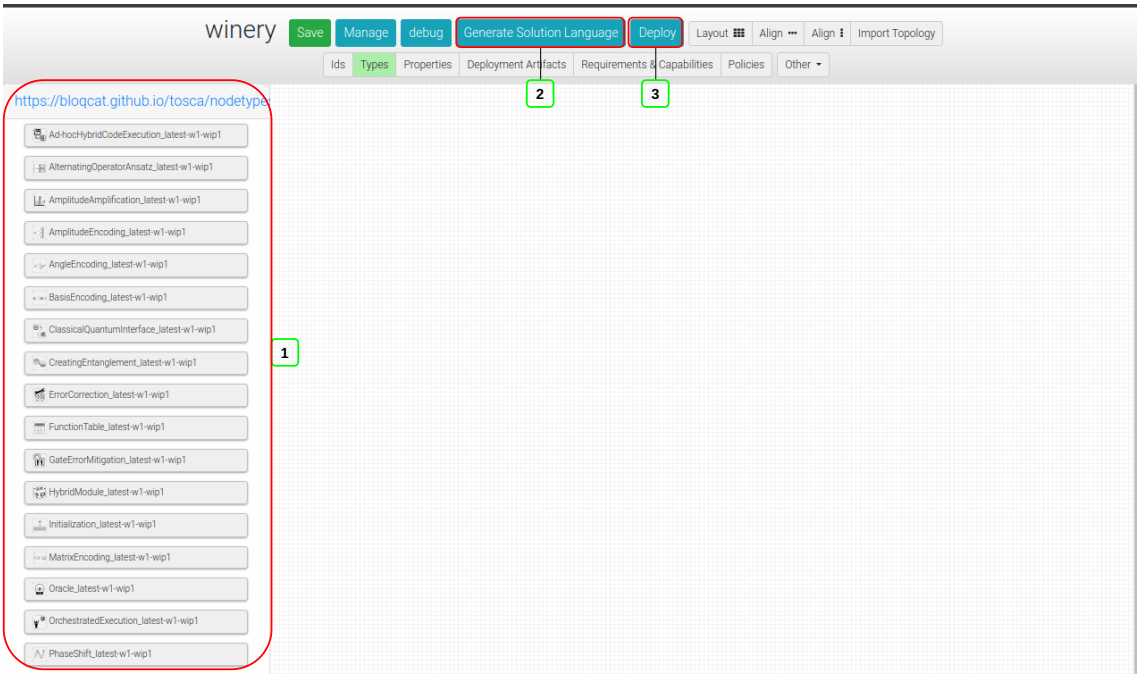

Abbildung 6.4: Screenshot der UI von Winery [Con23] Topologiemodeller für ein konkretes „Service Template“. (1) QC-Muster Knoten, welche im Topologiemodell verwendet werden können. (2) Button um eine Lösungssprache für die Musterspache zu generieren. (3) Sendet ein Rquest mit dem Topologiemodell im Request-Body für die Aggreagtion an das Aggregation Service [Ahm23].

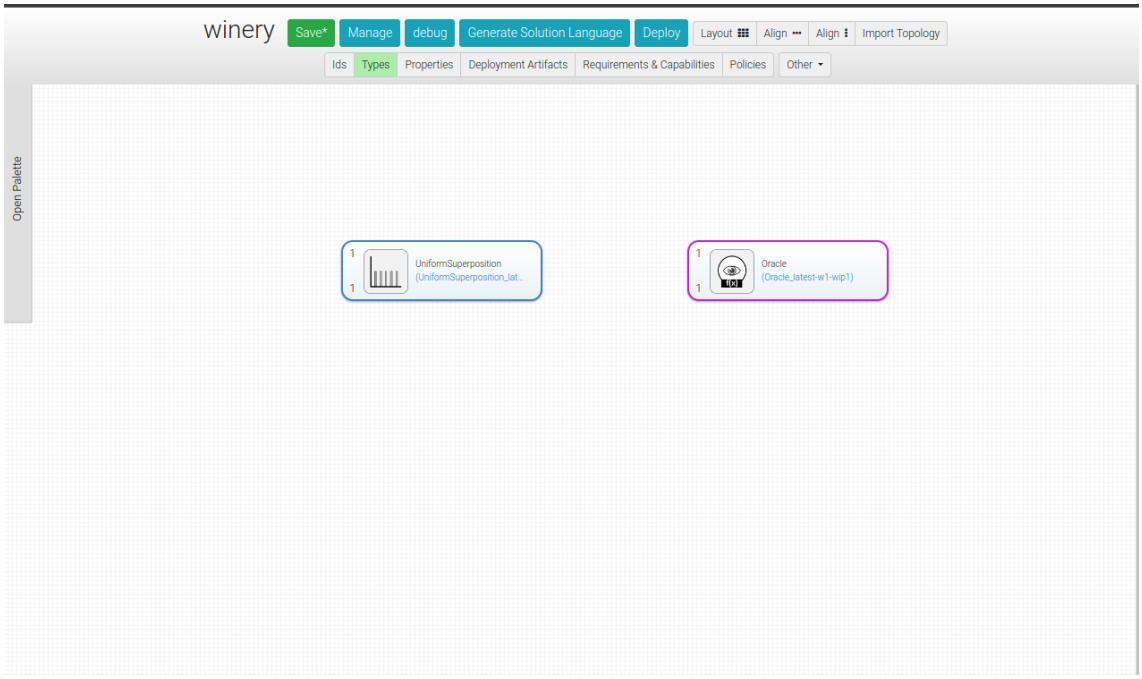

Abbildung 6.5: Screenshot von Winery [Con23] Topologiemodeller UI mit den Mustern *Uniform Superposition* und *Oracle*.

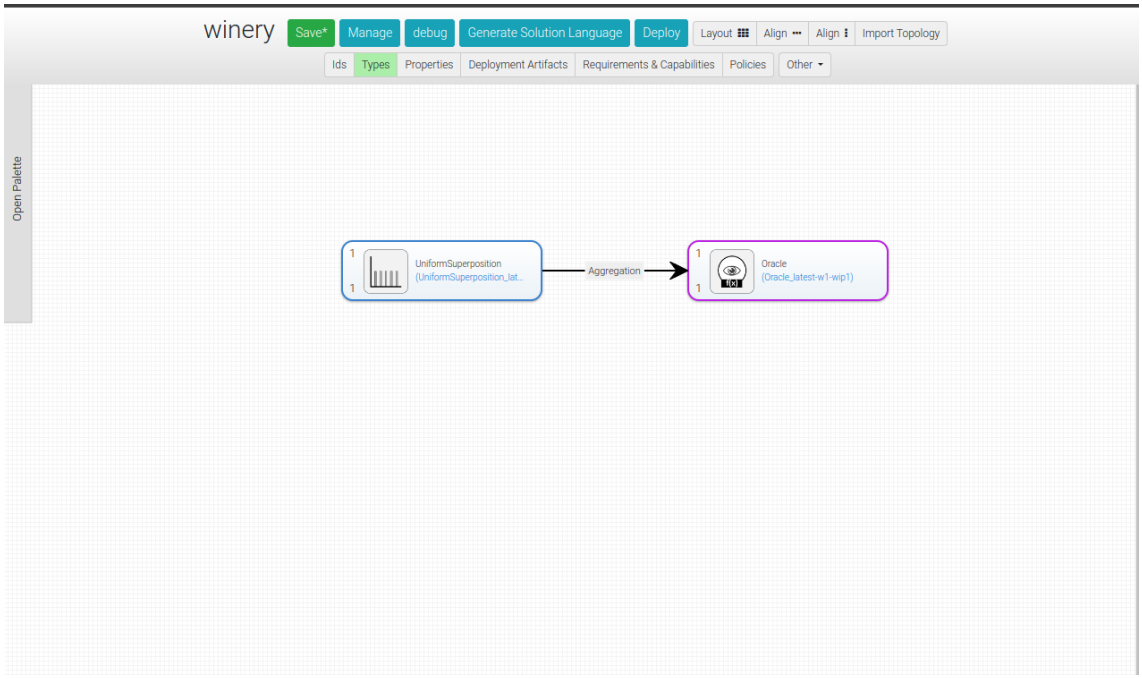

Abbildung 6.6: Screenshot von Winery [Con23] Topologiemodeller UI mit den Mustern *Uniform Superposition* und *Oracle*. Verbunden mit einer Relationen „Aggregation“

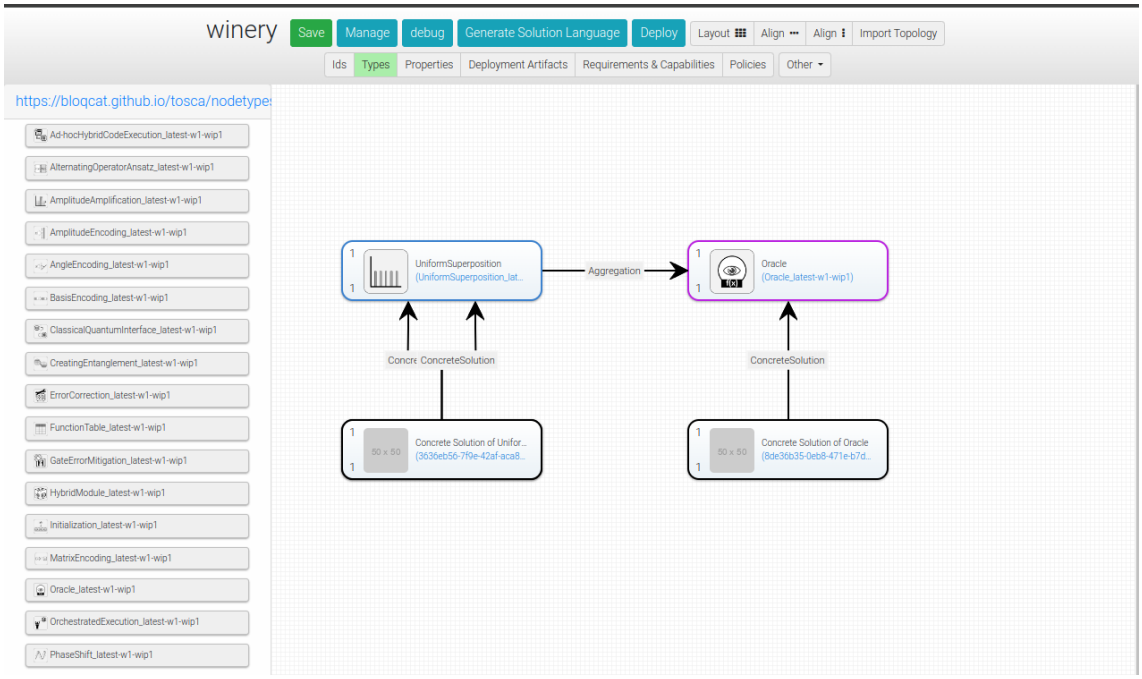

Abbildung 6.7: Screenshot von Winery [Con23] Topologiemodeller UI mit den Mustern *Uniform Superposition* und *Oracle*. Nachdem eine Lösungssprache generiert wurde

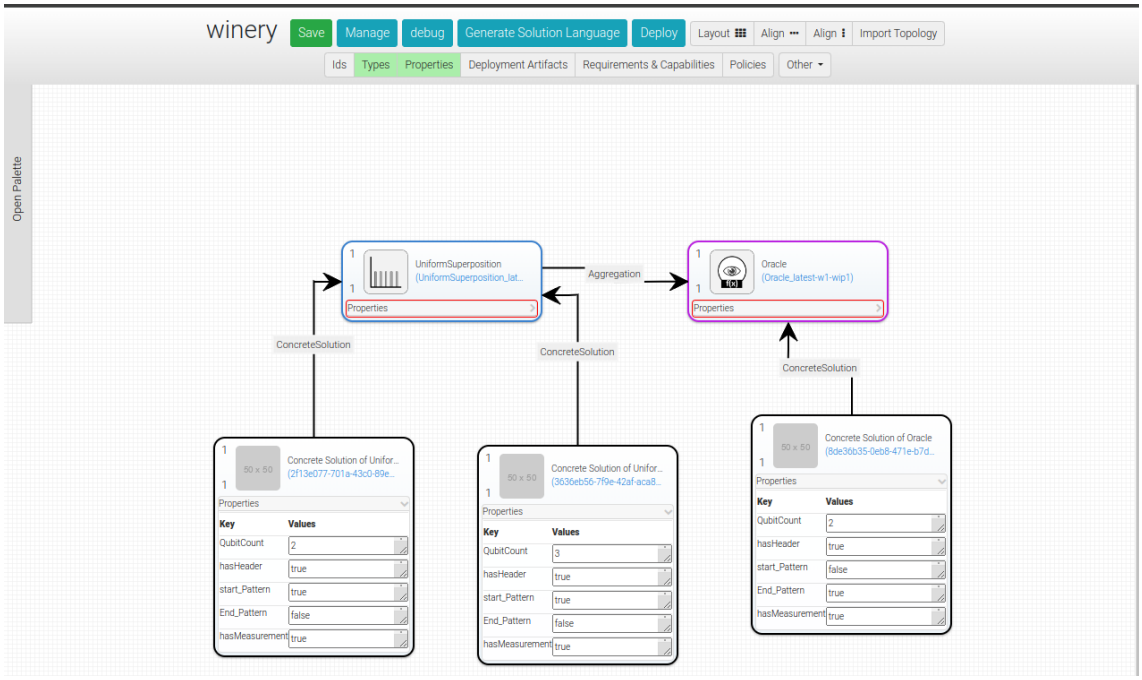

Abbildung 6.8: Screenshot von Winery [Con23] Topologiemodeller UI mit den Mustern *Uniform Superposition* und *Oracle*. Nachdem eine Lösungssprache generiert wurde. Die Metadaten der konkreten Lösungen sind sichtbar.

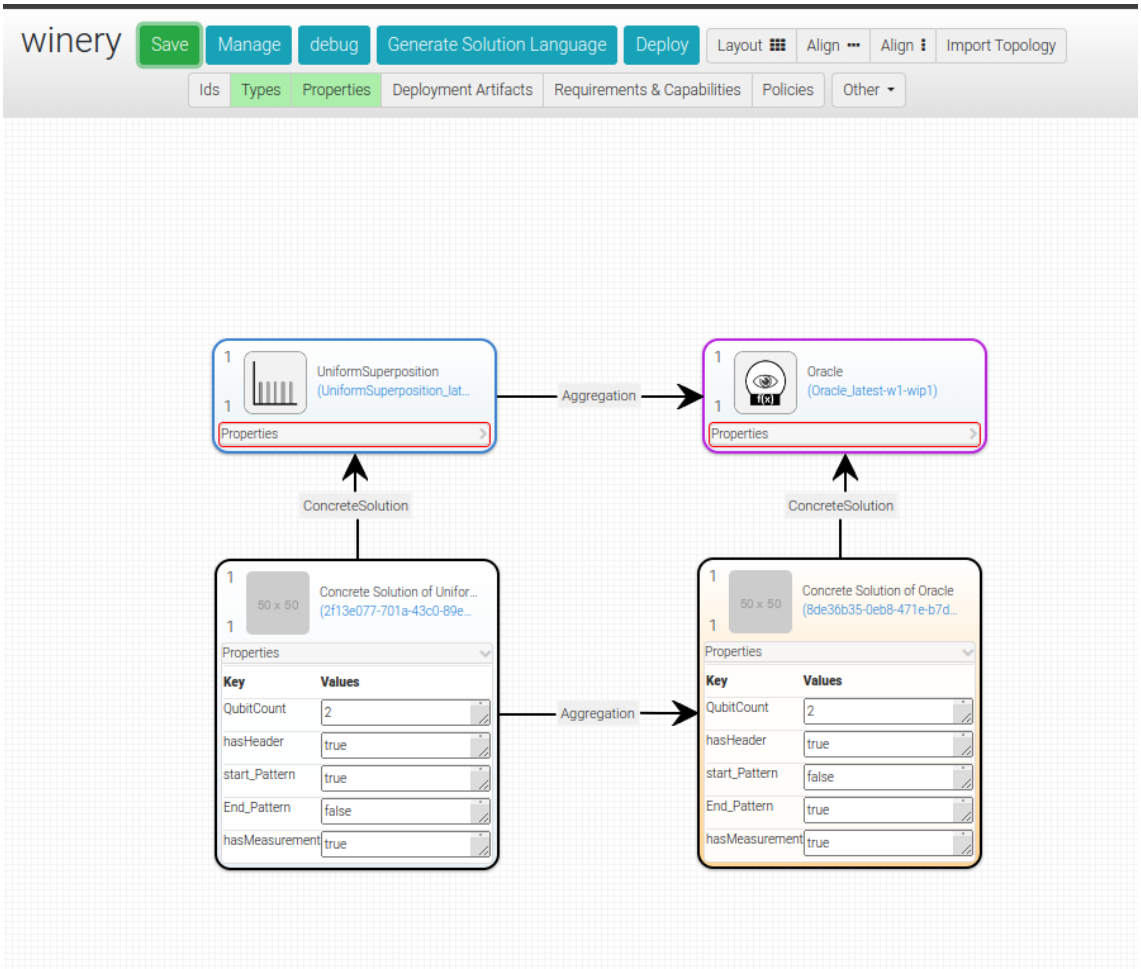

Abbildung 6.9: Screenshot von Winery [Con23] Topologiemodeller UI mit den Mustern *Uniform Superposition* und *Oracle*. Nachdem eine Lösungssprache generiert wurde und spezifische konkreten Lösungen für jedes Muster ausgewählt wurden und miteinander verbunden wurden.

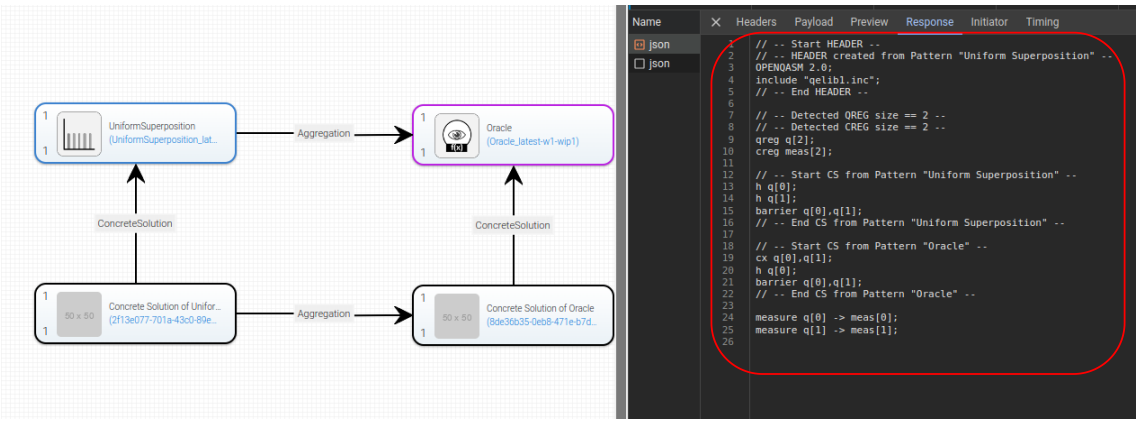

Abbildung 6.10: Screenshot von Webbrowser mit der Antwort des Aggregation Service [Ahm23] auf den „Deploy“-Request für das Deutsch-Algorithmus Topologiemodell.

7 Evaluation

Dieses Kapitel beinhaltet die Evaluation des in dieser Arbeit entwickelten Konzepts und der darauf basierten Implementierung, wobei sowohl technische Aspekte als auch die praktische Anwendbarkeit und die Erfüllung der definierten Anforderungen des Frameworks berücksichtigt werden.

7.1 Evaluation des Konzepts und der Implementierung

Die Evaluation des vorgestellten Frameworks, bestehend aus Konzept (siehe Kapitel 4) und Implementierung (siehe Kapitel 5), offenbart eine facettenreiche Mischung aus technischen Stärken und Schwächen. Eine der Hauptstärken liegt in der effizienten Kombination von bereits vorhandenen Open-Source-Komponenten mit neu entwickelten Elementen. Die Erweiterung von QC Atlas [QuA] und die Nutzung von Winery [Con23] sowie die Einführung der neuen Aggregation Service API [Ahm23] demonstrieren eine kluge Ressourcennutzung und Innovationsfähigkeit. Durch die Verwendung modularer Komponenten, wie der QC Atlas API und einer relationalen Datenbank, gewährleistet das Konzept eine hohe Flexibilität. Diese modulare Struktur ermöglicht es, einzelne Teile leicht anzupassen oder zu erweitern, was zur langfristigen Skalierbarkeit des Frameworks beiträgt. Der Einsatz von aktuellen Technologien wie Angular [Tea] für die Webanwendungen, Java für die APIs und SQL für die Datenbank zeigt, dass das Konzept auf modernen und bewährten Technologien basiert. Dies erhöht die Zuverlässigkeit und die Kompatibilität mit bestehenden Systemen. Zudem ermöglicht die Verwendung von Winery zur Modellierung eine graphbasierte, intuitive Darstellung komplexer Strukturen, was die Benutzerfreundlichkeit und Verständlichkeit wesentlich verbessert. Darüber hinaus erlauben die Erweiterungen an dem QC Atlas [QuA] das Hinterlegen konkreter Lösungen. Diese können mit bestimmten Mustern verbunden und an einem zentralen Speicherort hinterlegt werden, was eine effiziente Möglichkeit bietet, dieses Wissen zu speichern und wiederzuverwenden.

Trotz der Stärken des Frameworks gibt es Bereiche, die weiterentwickelt werden könnten. Ein zentrales Element, die neu entwickelte Aggregation Service API [Ahm23], ist noch nicht vollständig ausgereift. Derzeit unterstützt sie lediglich OpenQASM-2 als Datentyp für konkrete Lösungen. Eine Erweiterung des Aggregation Service, um beispielsweise direkten Datenbankzugriff oder fortgeschrittene Aggregationsfunktionen zu ermöglichen, würde die Vielseitigkeit und Effektivität des Frameworks signifikant steigern. Darüber hinaus besteht ein Optimierungsbedarf in der Handhabung mehrerer Circuits innerhalb einer einzigen OpenQASM-2-Datei. Die aktuelle Implementierung ist nicht auf die effiziente Verarbeitung mehrerer Circuits ausgelegt, was zu Herausforderungen bei der Skalierung und Flexibilität führt. Ebenso begrenzt die Unfähigkeit, Messungen innerhalb eines Circuits durchzuführen, die Einsatzmöglichkeiten des Frameworks in dynamischen Anwendungsfällen. Ein weiterer kritischer Punkt ist die fehlende Unterstützung von Quantensimulatoren. Die Integration eines Simulators würde nicht nur die Anwendbarkeit des Frameworks in einem breiteren Kontext ermöglichen, sondern auch das Testen verschiedener

Szenarien ohne die Notwendigkeit realer Quantum-Hardware erlauben. Zuletzt erfordert die Analyse und Verarbeitung der zu aggregierenden konkreten Lösungen eine Verbesserung. Aktuell können Namenskonflikte ein Hindernis darstellen. Eine präzisere und effizientere Analyse würde die Qualität der resultierenden Lösungen erhöhen. Dieser Aspekt ist besonders wichtig, da er direkt die Nutzbarkeit und Zuverlässigkeit der im Framework gesammelten Daten beeinflusst. Zukünftige Entwicklungen sollten daher auch auf verbesserte Analysemethoden abzielen, um die Effizienz des Frameworks weiter zu steigern.

Zusammenfassend lässt sich sagen, dass das Framework eine solide Basis für weitere Entwicklungen bietet und bereits in seinem aktuellen Zustand eine Reihe von Stärken aufweist, die es zu einem wertvollen Werkzeug in seinem Anwendungsbereich machen. Zukünftige Verbesserungen, insbesondere in Bezug auf die Unterstützung verschiedener Datentypen und die Erweiterung an die Aggregationsmöglichkeiten, könnten die Leistungsfähigkeit und Anwendungsbreite des Frameworks weiter erhöhen.

7.2 Evaluation der Anforderungen

Im Rahmen der Evaluation des Frameworks bezogen auf das Anwendungsszenario, wie in Abschnitt 6.1 beschrieben, wird nun die Erfüllung der in Abschnitt 3.2 definierten Anforderungen diskutiert. Hierbei wird jede einzelne Anforderung spezifisch anhand des genannten Anwendungsszenarios überprüft und bewertet.

A1 Auswahl von Mustern bzw. konkreten Lösungen soll mithilfe einer grafischen UI ermöglicht werden.

Die Erfüllung dieser Anforderung wurde durch den Einsatz des Winery Topologiemodellers, einem integralen Bestandteil von Winery [Con23], realisiert. Dieses Tool erlaubt es, Muster und konkrete Lösungen grafisch darzustellen und auszuwählen. Gemäß der in [Aldb] definierten Knotenstruktur für Muster ermöglicht der Winery Topologiemodeller eine direkte Auswahl dieser Muster. Auch die Auswahl der konkreten Lösungen erfolgt über denselben Modeller. Diese Funktionen tragen maßgeblich zu einer intuitiven und benutzerfreundlichen Interaktion bei, indem sie die Auswahl und Zuordnung von Mustern sowie konkreten Lösungen erleichtern und somit eine benutzerzentrierte Umgebung schaffen.

A2 Die konkreten Lösungen der ausgewählten Muster sollen zu einer Gesamtlösung aggregiert werden.

Die Umsetzung dieser Anforderung wurde durch die Implementierung der Aggregation Service API [Ahm23] realisiert. Diese API ermöglicht es, ausgewählten konkreten Lösungen, die miteinander in Beziehung stehen, zu einer Gesamtlösung zu aggregieren. Der Aggregationsprozess umfasst die Validierung, Analyse und das Zusammenführen der ausgewählten konkreten Lösungen. Durch diese Funktionalität können Benutzer effizient eine kohärente Gesamtlösung aus individuellen Entwurf konstruieren.

A3 Die aggregierte Gesamtlösung soll lauffähig sein und den Architekturentwurf implementieren.

Die Erfüllung dieser Anforderung wird durch manuelles Testen der aggregierten Gesamtlösungen überprüft, wobei ein Quantencomputer oder ein Simulator, wie zum Beispiel Qiskit [IBM], zum Einsatz kommt. Die in Abschnitt 6.1 beschriebene aggregierte Gesamtlösung wurde erfolgreich auf Qiskit [IBM] ausgeführt. Dieser Test demonstriert, dass die aggregierten Gesamtlösungen funktionsfähig sind und den vorgesehenen Architekturentwurf korrekt implementieren.

7.3 Zusammenfassung der Evaluation

Die Evaluation des entwickelten Frameworks offenbart eine effiziente Kombination aus bestehenden Open-Source-Komponenten und innovativen Neuentwicklungen. Besonders hervorzuheben sind die Flexibilität durch die modulare Struktur und die Benutzerfreundlichkeit, unterstützt durch den Einsatz moderner Technologien sowie die intuitive Darstellung durch Winery [Con23]. Die Erweiterungen des QC Atlas [QuA] tragen zur effektiven Speicherung und Wiederverwendung von Lösungsmustern bei.

Das Framework zeigt das Potenzial für weitere Entwicklungen. Eine solche Verbesserungsmöglichkeit betrifft die Aggregation Service API [Ahm23], die derzeit noch in der Entwicklung ist und momentan nur eine limitierte Anzahl von Datentypen unterstützt. Eine wichtige Erweiterungsmöglichkeit wäre die Integration eines Quantensimulators, wie beispielsweise Qiskit [IBM]. Diese Integration kann den direkten Versand der aggregierten Gesamtlösung an den Simulator ermöglichen, etwa über einen speziellen Button im Winery Topologiemodeller. Eine solche Funktion würde den Mehrwert des Frameworks deutlich erhöhen, indem sie den Benutzern eine direkte und nahtlose Interaktion mit Quantensimulationswerkzeugen ermöglicht und somit den praktischen Anwendungsbereich des Frameworks erweitert.

In der Gesamtbetrachtung erfüllt das Framework die gestellten Anforderungen erfolgreich. Die Möglichkeit zur Auswahl und Aggregation von Mustern und konkreten Lösungen durch den Winery Topologiemodeller [Con23] und die Aggregation Service API [Con23] demonstriert die Praxistauglichkeit und Anwendbarkeit des Frameworks. Zukünftige Verbesserungen, besonders in Bezug auf die Unterstützung verschiedener Datentypen und erweiterte Aggregationsfunktionen, könnten das Framework weiter stärken und seinen Anwendungsbereich erweitern.

8 Zusammenfassung und Ausblick

Diese Arbeit schließt mit einem Fazit und bietet einen Ausblick auf zukünftige Forschungsmöglichkeiten im Feld der automatisierten Aggregation von konkreten Lösungen sowie deren Speicherung für die Wiederverwendung im Bereich des QC.

8.1 Fazit

Ähnlich wie in anderen Technologiebereichen hat sich auch im Bereich des QC die Verwendung von Mustern zur Beschreibung wiederkehrender Probleme und deren bewährter, abstrakter Lösungen etabliert. Trotzdem werden diese Muster für neue Anwendungsfälle oft erneut implementiert. Vor diesem Hintergrund zielt diese Arbeit darauf ab, erneute Implementierungen zu vermeiden und die Wiederverwendung bereits implementierter Lösungen zu erleichtern. Durch die Implementierung des Frameworks wurde demonstriert, dass die Aggregation und Speicherung konkreter Lösungen im Bereich des Quantencomputing (QC) einen Mehrwert bieten, insbesondere für Benutzer, die mit dem QC nicht vertraut sind. Das entwickelte Framework bildet einen ersten Schritt in Richtung der Automatisierung der Speicherung und Aggregation von konkreten Lösungen, was manuelle Arbeit reduziert und die Effizienz sowie Effektivität im Bereich des QC steigern könnte. Trotz der erzielten Fortschritte sind auch Herausforderungen und Limitationen aufgetreten, insbesondere bei Entscheidungen über die sinnvolle Speicherung von konkreten Lösungen und den dazugehörigen Metadaten. Zudem erhöht die Vielfalt der Aggregationsmöglichkeiten für diese Lösungen die Komplexität zusätzlich. Trotz dieser Herausforderungen bietet unsere Arbeit ein solides Fundament für die Weiterentwicklung eines Frameworks in diesem dynamischen und schnell fortschreitenden Bereich.

8.2 Ausblick

Im Folgenden werden Anknüpfungspunkte an diese Arbeit vorgestellt und ein Ausblick auf weitere Entwicklungen des Frameworks gegeben.

Die vorliegende Arbeit eröffnet vielfältige Möglichkeiten für zukünftige Forschungen. Ein Hauptaugenmerk liegt auf der weiteren Entwicklung und Verfeinerung der Methoden zur Automatisierung der Aggregation und Speicherung von konkreten Lösungen. Ein deutlicher Bedarf besteht für fortgeschrittene Algorithmen und Techniken, insbesondere unter Berücksichtigung der wachsenden Menge an Lösungen. Der Einsatz von Techniken wie maschinellem Lernen und künstlicher Intelligenz könnte hierbei von großem Nutzen sein. Diese könnten nicht nur die automatische Generierung und Bewertung von Metadaten unterstützen, sondern auch die Qualität der aggregierten Lösungen insgesamt verbessern.

Darüber hinaus besteht ein weiterer wichtiger Forschungsbedarf in der Entwicklung benutzerfreundlicherer Schnittstellen. Diese sollten so gestaltet sein, dass Fehler von Benutzerseite aus minimiert und bei Bedarf entsprechende Rückmeldungen gegeben werden. Dies würde die Zugänglichkeit und Effizienz des Frameworks weiter erhöhen und einen zusätzlichen Mehrwert schaffen.

Ein weiteres zentrales Forschungsthema ist die Entwicklung eines Algorithmus zur Aggregation von heterogenen konkreten Lösungen. Ein solcher Algorithmus könnte die Integration von Lösungen, die in verschiedenen Programmiersprachen implementiert sind, erleichtern. Dadurch würde nicht nur die Anzahl der integrierbaren Lösungen erhöht, sondern auch die Gesamteffizienz des Frameworks gesteigert. Diese Entwicklung würde einen bedeutenden Schritt in Richtung einer umfassenderen und flexibleren Lösungsintegration im Bereich des QC darstellen.

8.3 Schlusswort

Diese Arbeit repräsentiert einen Schritt in der Erforschung von Mustern und ihrer Anwendung im QC. Es sollte sowohl theoretische als auch praktische Perspektiven vereinen, um einen umfassenden Einblick in dieses Feld zu gewähren. An dieser Stelle möchte ich meinem Betreuer, Daniel Georg, meinen herzlichsten Dank für seine hervorragende Unterstützung und wertvollen Anregungen aussprechen.

Literaturverzeichnis

[Ahm23] D. G. und Ahmed Ebrahim Aldekal. *BloQCat Service*. 14. Juli 2023. URL: https://github.com/GeorgDaniel/bloqCat (zitiert auf S. 31, 39, 44, 47, 49–51).

[Alda] A. E. Aldekal. *BloQCat Framework*. URL: https://github.com/aldekal/bloqcat-framework (zitiert auf S. 31, 41).

[Aldb] A. E. Aldekal. *BloQCat Modeling*. URL: https://github.com/aldekal/bloqCat-modeling (zitiert auf S. 31, 33, 34, 38, 42, 50).

[Aldc] A. E. Aldekal. *QC Atlas Content*. URL: https://github.com/aldekal/qc-atlas-content (zitiert auf S. 33).

[Ale77] C. Alexander. *A pattern language: towns, buildings, construction*. Oxford university press, 1977 (zitiert auf S. 17, 18, 21, 23).

[BBL+22] M. Beisel, J. Barzen, F. Leymann, F. Truger, B. Weder, V. Yussupov. „Patterns for quantum error handling“. In: *Proceedings of the 14th International Conference on Pervasive Patterns and Applications (PATTERNS 2022)*. Xpert Publishing Services (XPS) Wilmington, DE, USA. 2022, S. 22–30 (zitiert auf S. 25).

[BWP+17] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, S. Lloyd. „Quantum machine learning“. In: *Nature* 549.7671 (2017), S. 195–202 (zitiert auf S. 23).

[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. 26, 31, 39, 40).

[Cona] P. A. Contributors. *Pattern Atlas*. URL: https://github.com/PatternAtlas (zitiert auf S. 25).

[Conb] R. Contributors. *OpenQASM*. URL: https://github.com/openqasm/openqasm (zitiert auf S. 26).

[Con23] E. F. W. Contributors. *Winery*. 14. Juli 2023. URL: https://github.com/OpenTOSCA/winery (zitiert auf S. 18, 31, 34, 38, 41–47, 49–51).

[FBB+14] M. Falkenthal, J. Barzen, U. Breitenbüher, C. Fehling, F. Leymann. „From Pattern Languages to Solution Implementations“. In: *Proceedings of the 6 International Conference on Pervasive Patterns and Applications*. Xpert Publishing Services (XPS), 2014, S. 12–21 (zitiert auf S. 18, 21, 23).

[FBB+16] M. Falkenthal, J. Barzen, U. Breitenbücher, C. Fehling, F. Leymann, A. Hadjakos, F. Hentschel, H. Schulze. „Leveraging Pattern Application via Pattern Refinement“. In: *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*. epubli GmbH, 2016, S. 38–61 (zitiert auf S. 18, 21).

[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&engl=0 (zitiert auf S. 17, 21–23).

[FBBL17] M. Falkenthal, J. Barzen, U. Breitenbüucher, F. Leymann. „Solution Languages: Easing Pattern Composition in Different Domains“. In: *International Journal on Advances in Software* (2017), S. 263–274 (zitiert auf S. 17, 18, 21, 24).

[FBBL19] M. Falkenthal, U. Breitenbücher, J. Barzen, F. Leymann. „On the algebraic properties of concrete solution aggregation“. Englisch. In: *SICS Software-Intensive Cyber-Physical Systems* (Feb. 2019), S. 1–12. URL: http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=ART-2019-04&engl=0 (zitiert auf S. 24, 27, 29).

[FBFL15] C. Fehling, J. Barzen, M. Falkenthal, F. Leymann. „PatternPedia – Collabroative Pattern Identification and Authoring“. In: *Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC)*. epubli GmbH, Juni 2015 (zitiert auf S. 24, 25).

[FL17] M. Falkenthal, F. Leymann. „Easing Pattern Application by Means of Solution Languages“. In: *Proceedings of the 9th International Conference on Pervasive Patterns and Applications*. Xpert Publishing Services (XPS), 2017, S. 58–64 (zitiert auf S. 17, 18, 21).

[GBB+23] D. Georg., J. Barzen., M. Beisel., F. Leymann., J. Obst., D. Vietz., B. Weder., V. Yussupov. „Execution Patterns for Quantum Applications“. In: *Proceedings of the 18th International Conference on Software Technologies - ICSOFT*. INSTICC. SciTePress, 2023, S. 258–268. ISBN: 978-989-758-665-1. DOI: 10.5220/0012057700003538 (zitiert auf S. 25).

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. Aufl. Addison-Wesley Professional, 1994. ISBN: 0201633612. URL: http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1 (zitiert auf S. 17, 22).

[Gro96] L. K. Grover. „A Fast quantum mechanical algorithm for database search“. In: (Mai 1996). arXiv: quant-ph/9605043 (zitiert auf S. 23).

[Hom08] M. Homeister. *Quantum Computing verstehen*. Springer, 2008 (zitiert auf S. 41).

[HW02] G. Hohpe, B. Woolf. „Enterprise integration patterns“. In: *9th conference on pattern language of programs*. 2002, S. 1–9 (zitiert auf S. 17, 21).

[IBM] IBM. *Qiskit*. URL: https://github.com/Qiskit/qiskit (zitiert auf S. 51).

[Inca] D. Inc. *Docker*. URL: https://www.docker.com/ (zitiert auf S. 31).

[Incb] G. Inc. *GitHub*. URL: https://github.com/ (zitiert auf S. 31, 33).

[KBBL13] O. Kopp, T. Binz, U. Breitenbücher, F. Leymann. „Winery – Modeling Tool for TOSCA-based Cloud Applications“. In: *11th International Conference on Service-Oriented Computing*. LNCS. Springer, 2013 (zitiert auf S. 31, 35).

[LB21] F. Leymann, J. Barzen. „Pattern Atlas“. In: *Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future*. Hrsg. von M. Aiello, A. Bouguettaya, D. A. Tamburri, W.-J. van den Heuvel. Cham: Springer International Publishing, 2021, S. 67–76. ISBN: 978-3-030-73203-5. DOI: 10.1007/978-3-030-73203-5_5. URL: https://doi.org/10.1007/978-3-030-73203-5_5 (zitiert auf S. 25).

[Ley19] F. Leymann. „Towards a Pattern Language for Quantum Algorithms“. In: *Quantum Technology and Optimization Problems*. Bd. 11413. Lecture Notes in Computer Science (LNCS). Cham: Springer International Publishing, 2019, S. 218–230. DOI: 10.1007/978-3-030-14082-3_19 (zitiert auf S. 17, 21–23, 28).

[LJL+10] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, J. L. O’Brien. „Quantum computers“. In: *nature* 464.7285 (2010), S. 45–53 (zitiert auf S. 17).

[NC10] M. A. Nielsen, I. L. Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010 (zitiert auf S. 23).

[Pal] Pallets. *Flask*. URL: https://flask.palletsprojects.com/en/3.0.x/# (zitiert auf S. 39).

[Piv] Pivotal Software. *Spring Boot*. URL: https://spring.io/projects/spring-boot (zitiert auf S. 31).

[Pla] PlanQK. *Pattern Atlas*. URL: https://patterns.platform.planqk.de/pattern-languages (zitiert auf S. 18, 25, 28, 31, 33, 35, 37).

[Pos] PostgreSQL Global Development Group. *PostgreSQL*. URL: https://www.postgresql.org/ (zitiert auf S. 31).

[Pre18] J. Preskill. „Quantum computing in the NISQ era and beyond“. In: *Quantum* 2 (2018), S. 79 (zitiert auf S. 23).

[QuA] QuAntiL. *QC Atlas*. URL: https://github.com/UST-QuAntiL/qc-atlas (zitiert auf S. 31, 33, 35–38, 40–42, 49, 51).

[Rei12] R. Reiners. „A Pattern Evolution Process-From Ideas to Patterns.“ In: *Informatiktage*. 2012, S. 115–118 (zitiert auf S. 17).

[Sho94] P. Shor. „Algorithms for quantum computation: discrete logarithms and factoring“. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, S. 124–134. DOI: 10.1109/SFCS.1994.365700 (zitiert auf S. 23).

[Tea] A. Team. *Angular Framework*. URL: https://angular.io/ (zitiert auf S. 31, 49).

[WBLS21a] M. Weigold, J. Barzen, F. Leymann, M. Salm. „Encoding patterns for quantum algorithms“. In: *IET Quantum Communication* 2.4 (2021), S. 141–152. DOI: https://doi.org/10.1049/qtc2.12032. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/qtc2.12032. URL: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/qtc2.12032 (zitiert auf S. 25).

[WBLS21b] M. Weigold, J. Barzen, F. Leymann, M. Salm. „Expanding Data Encoding Patterns For Quantum Algorithms“. In: *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*. 2021, S. 95–101. DOI: 10.1109/ICSA-C52384.2021.00025 (zitiert auf S. 25).

[WBLS22] M. Weigold, J. Barzen, F. Leymann, M. Salm. „Data Encoding Patterns for Quantum Computing“. In: *Proceedings of the 27th Conference on Pattern Languages of Programs*. PLoP '20. Virtual Event: The Hillside Group, 2022. ISBN: 9781941652169 (zitiert auf S. 25).

[WBLV21] M. Weigold, J. Barzen, F. Leymann, D. Vietz. „Patterns for Hybrid Quantum Algorithms“. In: *Service-Oriented Computing*. Hrsg. von J. Barzen. Cham: Springer International Publishing, 2021, S. 34–51. ISBN: 978-3-030-87568-8 (zitiert auf S. 25).

[Zal98] C. Zalka. „Efficient simulation of quantum systems by quantum computers“. In: *Fortschritte der Physik: Progress of Physics* 46.6-8 (1998), S. 877–879 (zitiert auf S. 23).

Alle URLs wurden zuletzt am 17. Januar 2024 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