An Analysis of Ontological Entities to Represent Knowledge on Quantum Computing Algorithms and Implementations

Darya Martyniuk[1], Michael Falkenthal[2], Naouel Karam[1], Adrian Paschke[1], and Karoline Wild[3]

[1] Data Analytics Center, Fraunhofer FOKUS, Berlin, Germany
{darya.martyniuk, naouel.karam, adrian.paschke}@fokus.fraunhofer.de
[2] StoneOne AG, Berlin, Germany
michael.falkenthal@stoneone.de
[3] Institute of Architecture of Application Systems, University of Stuttgart, Germany
karoline.wild@iaas.uni-stuttgart.de

Abstract. The field of quantum computing is developing rapidly. As a result, a variety of quantum hardware, software development kits, and quantum algorithms have been developed in recent years. However, knowledge about these artifacts is either not available or spread among different sources. Thus, to analyze, compare, and evaluate knowledge on quantum computing an integrated knowledge base is required. In this paper, we introduce key concepts of an ontology for quantum algorithms and their implementations. The presented ontology serves as basis for a collaborative platform for researchers and practitioners to support collection and development of knowledge on the field of quantum computing.

Keywords: Ontology, Taxonomy, Quantum Computing, Quantum Algorithm

1 Introduction and Motivation

The advent of publicly available quantum computers in recent years has boosted the developments of quantum software, quantum programming languages, and quantum platforms a lot. Quantum computers are no longer just scientific experiments but can be accessed even outside quantum hardware vendors' laboratories via application programming interfaces (API) and software development kits (SDK) [9]. SDKs provided by vendors such as IBM, Google, D-Wave, and Rigetti, include initial implementations of quantum algorithms, which act perfectly as training material for people trying to gain insights into the area of quantum information. These endeavors, combined with the huge efforts undertaken by the scientific community around the world to find new quantum algorithms that outperform known classical algorithms, have led to a manifold of publications and

exemplary programming code, which can form a valuable body of knowledge for the development of quantum-inspired and -enhanced applications in the future. This is important because, even being in the so-called Noisy Intermediate Scale Quantum (NISQ) era [19], quantum computing has the potential to become a disruptive innovation driver in many different fields, where we are facing the limitations of execution powers of classical computers. So, for example, it is very likely that quantum computing will allow to tackle complex problems from molecule simulations [7], the broad topic of artificial intelligence (AI) [22], and even the optimization of energy systems [2].

However, for companies and scientists trying to leverage quantum computing for business, development, and research, this means hard times ahead. This is because the knowledge about quantum computing and, especially, how to build applications that can make use of quantum resources to gain improvements over classical algorithms is not systematically available. Explanations of quantum algorithms, such as estimations of the speedup over classical algorithms, theoretical considerations of required quantum resources, and actual implementations of the algorithms are typically spread among different sources. As a consequence, required knowledge has to be collected and obtained manually from a vast amount of scientific publications, vendor-specific documentation pages, or public code repositories. Yet, such information has to be analyzed holistically to understand how to implement quantum algorithms and quantum applications for specific use cases and scientific problems at hand. Moreover, the absence of an integrated knowledge base hinders to establish a community bridging theoretical foundations and research on new quantum algorithms with their usage, application, and implementation for relevant and real scenarios. In this regard, an open ecosystem is key for rapid technological progress in quantum computing via close cooperation and steady interchange of ideas between research and industry [16].

Therefore, it is required to come up with a systematic approach to semantically represent and curate knowledge about quantum computing algorithms along with their implementations, as presented in this paper. We contribute with a semantified meta-model that was engineered to provide the core knowledge structure for semantic knowledge curation in the project *PlanQK – platform and ecosystem for quantum-enhanced AI* [13, 18]. The semantic curation of knowledge artifacts described and managed on the PlanQK platform provides the basis for e.g. semantic search and semantic service functionalities. PlanQK aims to lay a body of knowledge to the field of quantum-enhanced AI via a publicly accessible platform following the mindset already taken in classical AI [23]. Besides the pure capturing of quantum algorithms and their implementations it is also inevitable to establish a community-based discussion and exchange around the captured knowledge to enable continuous evolution of the knowledge.

This paper is structured as following: We introduce the PlanQK project in Sect. 2 and discuss the key objectives of the intended knowledge platform. We introduce the mentioned core meta-model as an ontology in Sect. 3. Finally, we conclude this paper in Sect. 4 by an outlook to future work, which will be conducted in PlanQK based on the presented ontology.

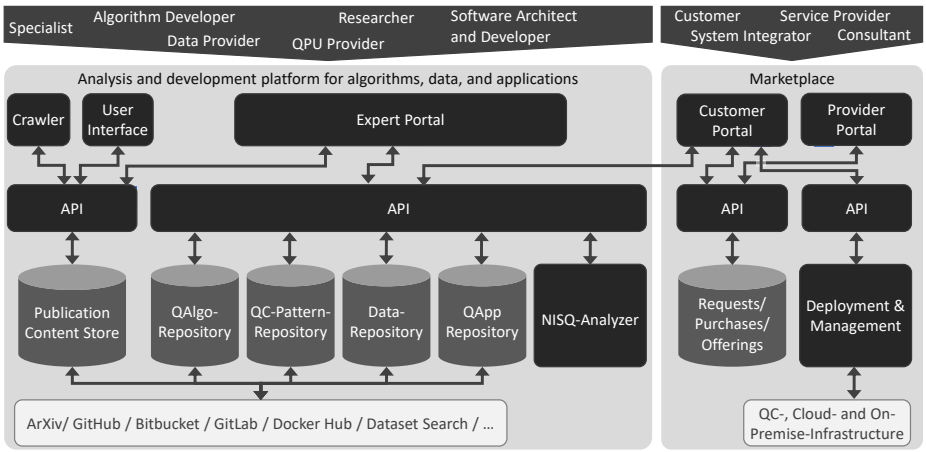

Fig. 1: Architecture of the PlanQK Platform [12].

2 The PlanQK Project

The vision of a collaborative platform for the exchange of knowledge in the field of quantum computing has already been presented in previous work [11, 12]. The PlanQK project will realize this vision by providing a platform that enables (i) knowledge and technology transfer between research and industry, (ii) a vendor-agnostic access to quantum computing resources, and (iii) quantum applications as a service. Fig. 1 depicts the PlanQK architecture with the two main components: The *analysis and development platform* and the *marketplace*. The goal of the first component is to provide a platform for quantum experts to collect, discuss, evaluate, and share knowledge, e. g., publications, algorithms, or implementations. Marketplace, on the other hand, offers solutions in form of quantum applications and consulting services to users with a specific problem to be solved.

The key knowledge artifacts that are provided on the analysis and development platform include *Quantum Algorithms (QAlgos)* and their *Implementations* using different SDKs for the different quantum computing vendors, *Quantum Design Patterns (QC-Patterns)* that provide best practices for quantum algorithms, *Datasets (Data)* for specific machine learning and AI algorithm, and *Quantum Applications (QApps)* that can be deployed and integrated with classical applications. New knowledge, e. g., in form of publications, can be added using the *user interface* or in an automated manner using the *crawler*. The knowledge artifacts can then be extracted and linked, discussed and evaluated, and made available to customers to identify suitable algorithms or implementations for their specific use cases. Selected QApps can be automatically deployed and managed, e. g., using established deployment technologies. The *NISQ-Analyzer* component supports users by analyzing which implementation of a quantum algorithm and which quantum computer are recommended for specific input data [21].

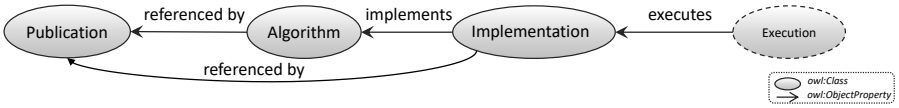

Fig. 2: Central concepts of the PlanQK ontology.

An important goal of the PlanQK platform is to build a quantum community and establish knowledge exchange between scientific and commercial users. The ontology presented in this work defines an unambiguous machine-interpretable semantic description of the platform knowledge artifacts and intend to used to support the user exchange through (i) proving the logical consistency of the curated data, (ii) serving as the foundation for the realization of various AI-based features, e.g., semantic faceted search, natural language question answering engine, or recommendation system for the selection of an appropriate quantum algorithm in a specific use case, and (iii) providing an opportunity to document the expert discussions held on the platform in a semantically usable way.

3 Towards a Unified Ontology for Quantum Algorithms

In developing the ontology, we followed existing ontology development guidelines [4, 5, 17]. As a first step towards the knowledge definition, we derived competency questions from general platform requirements specified by potential users (both quantum computing researchers and industry partners). Based on the list of competency questions, we identified key concepts related to the description of quantum algorithms and their implementations. The ontology draft has been then presented to quantum computing experts to collect their feedback and has been improved accordingly.

Fig. 2 depicts the connections between key concepts of the ontology, which we specify as OWL[4] classes. This design was inspired by the Algorithm-Implementation-Execution Ontology Design Pattern (ODP) [10] and the ML-Schema Core Vocabulary [20]. The class *Algorithm* aggregates characteristics that are common for quantum and classical algorithms and *Implementation* represents a realization of an algorithm. *Publication* represents publications about an algorithm or an algorithm implementation. Since the class *Execution* requires in-depth knowledge about quantum hardware and input data, it will be considered in detail in future work. In the following, we introduce the classes *Algorithm* (Sect. 3.1) and *Implementation* (Sect. 3.2) with their related concepts in detail.

3.1 Algorithms

Fig. 3 shows selected object properties of the class *Algorithm*. An algorithm provides a solution for a specific problem. We formalize this using two classes,

[4] `https://www.w3.org/OWL/`

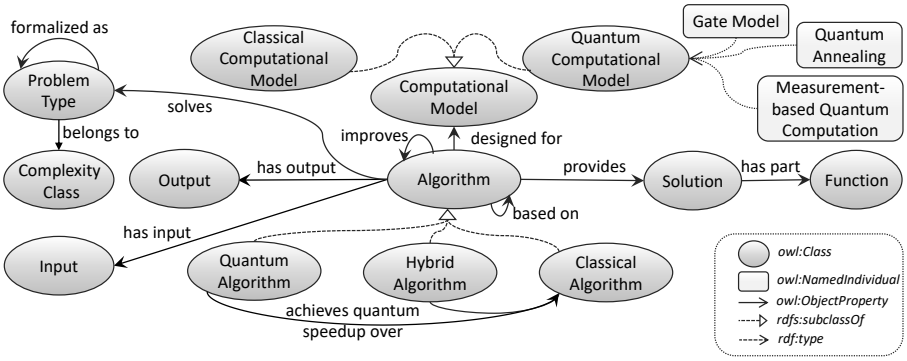

Fig. 3: Description of an algorithm in the PlanQK ontology.

Solution and *Problem Type*, and connect them with *Algorithm* through object properties *"provides"* and *"solves"*. *Solution* is defined as algorithmic steps to solve a computation problem, typically pseudocode or/and circuit. *Solution* can have connection to the class *Function* when it is an essential part of the solution, such as an appropriate objective function is a fundamental element of optimization algorithms or a quantum oracle is a crucial part of some quantum algorithms, e.g., Deutsch-Jozsa algorithm. *Problem Type* represents types of problems that are solvable by algorithms. Many real-world problems can be mathematically *formalized* in terms of other problems, e. g., a cluster problem can be expressed as an optimization problem. *Complexity Class* stores knowledge about the hardness of a problem. The class *Input* represents the input accepted by an algorithm, and *Output* specifies the output produced by an algorithm. The ontology is able to express two relation types between algorithms: An algorithm can *improve* an existing algorithm or be *based on* the idea of other algorithms.

A researcher designs an algorithm with a computational model in mind. We distinguish between *Classical* and *Quantum Computational Model* and classify algorithms in *Quantum*, *Classical* and *Hybrid Algorithms* depending on the model for which an algorithm was designed. *Quantum Algorithm* is an algorithm that is designed only for a quantum computational model. Examples of the *Quantum Computational Model* are *"gate model"*, *"measurement-based quantum computation"*, *"quantum annealing"*. *Classical Algorithm* is equivalent to an algorithm that is constructed only for a classical computational model. *Hybrid Algorithm* is an algorithm that utilizes some quantum and some classical computational models. In addition to the above presented classification of algorithms, the ontology also forms the taxonomy of algorithms based on the underlying problem type (machine learning, optimization, search algorithms etc.). The classes for specific algorithm types can act in the future as connection points for linking semantically similar knowledge bases, e. g., machine learning algorithms can be linked with such knowledge bases as MEX [3], a lightweight vocabulary for exchanging machine learning metadata, or ANNETT-O [6], an ontology for describing artificial neural network evaluation, topology, and training.

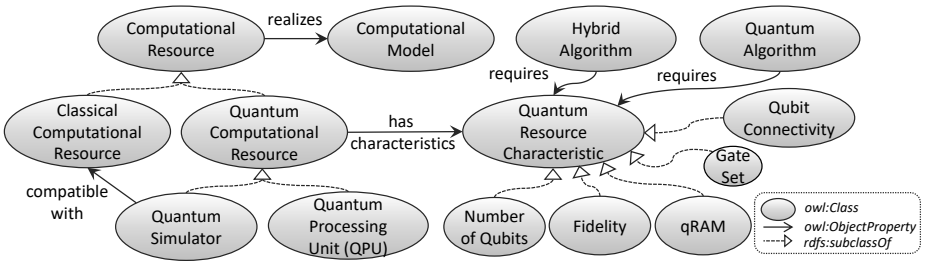

Fig. 4: Relations between *Computational Model* and *Computational Resource*.

By exploiting quantum-mechanical effects, such as superposition, entanglement, and quantum tunneling, quantum models perform computation more efficiently than classical ones [1]. Thus, quantum and hybrid algorithms can achieve speedup over their best known classical counterparts. Formalizing this consideration, we connect the classes *Quantum Algorithm* and *Hybrid Algorithm* through the property *"achieves quantum speedup over"* with the class *Classical Algorithm*.

For the class *Algorithm* we define the following data properties (not shown in Fig. 3): *"skos:prefLabel"* and *"skos:altLabel"* (algorithm name and its abbreviation, this properties are reused from SKOS Schema [14]), *"intend"* (the idea of the algorithm described in 1-2 sentences), *"assumption"* (basic assumptions underlying the algorithm), and *"limitation"* (known limitations of the algorithm). The classes *Quantum Algorithm* and *Hybrid Algorithm* have additional data properties: *"NISQ-ready"* from type *"xsd:boolean"*, which relates an expert estimation whether the execution of an algorithm on NISQ-devices will be possible, and *"expected quantum speedup"*, which specifies speedup (e. g., "exponential") obtained by the algorithm over the classical methods for the same task.

A *Computational Model* is realized by a *Computational Resource*, which collects instances utilized for the execution of an implementation. Fig. 4 illustrates relations between *Computational Model* and *Computational Resource*. We distinguish between *Classical* and *Quantum Computational Resource* that realize only *Classical* and only *Quantum Computational Model*, respectively. Quantum computational resources are categorized in *Quantum Processing Unit (QPU)* and *Quantum Simulator*. Since quantum simulators run on a classical hardware, *Quantum Simulator* is connected with *Classical Computational Resource* through the property *"compatible with"*. Quantum computational resources have specific characteristics defined by the class *Quantum Resource Characteristic*, which has subclasses *Gate Set*, *Fidelity*, *Quantum Access Random Memory (qRAM)*, *Number of Qubits*, and *Qubit Connectivity*. Also quantum and hybrid algorithms can specify hardware characteristics that are required for their realization, as shown in Fig. 4. For example, an algorithm can require a specific set of gates realizable only by some computational resources or show a theoretical speedup but assumes the availability of a qRAM such as Grover's algorithm for database search, or HHL for solving linear systems of equations. This information allows

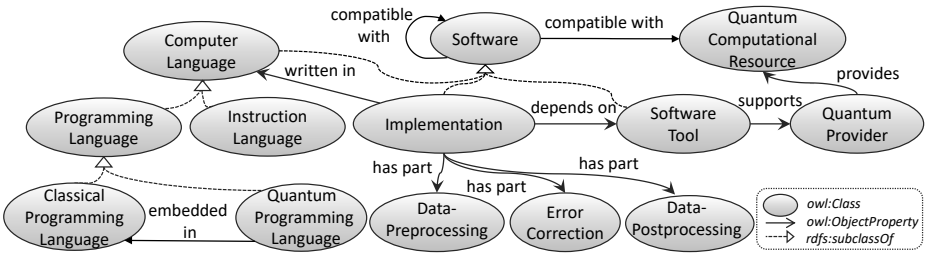

Fig. 5: Description of an algorithm implementation.

developers (i) to evaluate the practical usage of an algorithm, and (ii) to select an appropriate quantum computational resource for executing an algorithm.

3.2 Algorithm Implementation

Fig. 5 presents classes connected with the class *Implementation*. An implementation is *written in* a *Computer Language* and can *depend on* a *Software Tool*. The class *Computer Language* formalizes languages that can be used to write a machine or programming code. Similar to Lando et al. [8] and LaRose [9], we distinguish between *Instruction* and *Programming Language*. *Instruction Language* is defined as a low-level machine language used to instruct the computer which physical operation to perform on which bit [8]. *Instruction Language* has a subclass *Quantum Instruction Language* with individuals such as *"Quil"* and *"OpenQASM"*. *Programming Language* has subclasses *Classical* and *Quantum Programming Language* and contains languages that have a human readable syntax. Some quantum vendors provide quantum programming languages that are *embedded in* classical host languages, e. g., PyQuil from Forest SDK or Qiskit from Qiskit SDK are embedded in Python [9]. Others introduce new programming languages, such as Q# from Microsoft. *Software Tool* collects various frameworks, libraries, SDKs, or APIs that developers can use for implementing an algorithm. We specify classes *Implementation*, *Computer Language* and *Software Tool* as subclasses of *Software*. Guided by Computer System ODP [15], *Software* is connected with *Computational Resource* and with itself with the property *"compatible with"*.

The class *Quantum Provider* collects vendors that provide access to quantum computational resources. Most quantum providers supply software tools for developing and executing implementations on their quantum computational resources [9]. Analyzing various software tools, we came to the conclusion that if a software tool supports a specific quantum hardware provider, e. g., IBM, Rigetti, or D-Wave, it is compatible with all quantum computational resources provided by them. To formalize it, we connect the classes *Software Tool* and *Quantum Hardware Provider* through the property *"supports"* and define the relation *"compatible with"* as a superproperty of the chain *"supports"* and *"provides"*.

Apart from the algorithm realization itself, an implementation can include error correction, pre- and postprocessing. *Error Correction Technique* collects

methods for limiting errors that occur during information processing. *Data Preprocessing Technique* specifies operations that are applied on the input data before the actual algorithm steps will be executed. The class *Data Postprocessing Technique* includes methods that are applied on the algorithm output. A common postprocessing technique for quantum and hybrid algorithms is the *Read-out Error Correction* that is specified in the ontology as a subclass of both *Error Correction Technique* and *Data Postprocessing Technique*.

Ontology source can be found in our repository[5]. The first prototypical implementation of the platform services has shown that the ontology is consistent and can be used for the realization of semantic features.

4 Conclusion and Future Work

In this work we present the first key entities towards a unified ontology for semantically curating knowledge about quantum algorithms and their implementations. The ontology provides a basis for the AI-powered semantic features on the PlanQK platform, such as semantic search and semantic services. The next steps will be to extend the ontology to not yet covered knowledge artifacts, such as quantum applications and data, and to refine the existing artifacts. In addition, provenance data about implementation execution and performance will be included, which is important to identify suitable algorithms for a given problem. Furthermore, a standard-based API for supporting the semantic access to the curated knowledge artifacts and semantic services will be developed.

Acknowledgments This work was partially funded by the BMWi project *PlanQK (01MK20005N / 01MK20005F)*.

References

1. Abhijith, J., et al.: Quantum algorithm implementations for beginners. arXiv preprint arXiv:1804.03719 (2020)
2. Ajagekar, A., You, F.: Quantum computing for energy systems optimization: Challenges and opportunities. Energy **179**, 76–89 (2019)
3. Esteves, D., Moussallem, D., Baron, C., Soru, T., Usbeck, R., Ackermann, M., Lehmann, J.: Mex vocabulary: a lightweight interchange format for machine learning experiments. In: SEMANTICS '15 (2015)
4. Garijo, D., Poveda-Villalón, M.: Best practices for implementing fair vocabularies and ontologies on the web. ArXiv **abs/2003.13084** (2020)
5. Karapiperis, S., Apostolou, D.: Consensus building in collaborative ontology engineering processes. j-jukm **1**(3), 199–216 (dec 2006)
6. Klampanos, I.A., Davvetas, A., Koukourikos, A., Karkaletsis, V.: Annett-o: An ontology for describing artificial neural network evaluation, topology and training. Int. J. Metadata Semant. Ontologies **13**, 179–190 (2019)

[5] `https://github.com/PlanQK/semantic-services`

7. Kühn, M., Zanker, S., Deglmann, P., Marthaler, M., Weiß, H.: Accuracy and resource estimations for quantum chemistry on a near-term quantum computer. Journal of Chemical Theory and Computation **15**(9), 4764–4780 (2019)
8. Lando, P., Lapujade, A., Kassel, G., Fürst, F.: Towards a general ontology of computer programs. In: ICSOFT (2007)
9. LaRose, R.: Overview and Comparison of Gate Level Quantum Software Platforms. Quantum **3**, 130 (2019)
10. Lawrynowicz, A., Esteves, D., Panov, P., Soru, T., Dzeroski, S., Vanschoren, J.: An algorithm, implementation and execution ontology design pattern. In: Hammar, K., Hitzler, P., Krisnadhi, A., Lawrynowicz, A., Nuzzolese, A.G., Solanki, M. (eds.) Advances in Ontology Design and Patterns. Studies on the Semantic Web, vol. 32, pp. 55–68. IOS Press (2016)
11. Leymann, F., Barzen, J., Falkenthal, M.: Towards a Platform for Sharing Quantum Software. In: Proceedings of the 13$^{\text{th}}$ Advanced Summer School on Service Oriented Computing (2019). pp. 70–74. IBM Technical Report (RC25685), IBM Research Division (Sep 2019)
12. Leymann, F., Barzen, J., Falkenthal, M., Vietz, D., Weder, B., Wild, K.: Quantum in the Cloud: Application Potentials and Research Opportunities. In: Proceedings of the 10$^{\text{th}}$ International Conference on Cloud Computing and Services Science (CLOSER 2020). pp. 9–24. SciTePress (May 2020)
13. Linnhoff-Popien, C.: PlanQK — Quantum Computing Meets Artificial Intelligence. Digitale Welt **4**, 28–35 (2020)
14. Miles, A., Bechhofer, S.: SKOS simple knowledge organization system reference. Tech. rep., W3C (2009), `https://www.w3.org/TR/skos-reference/`
15. Mitzias, P., Kontopoulos, E., Riga, M.: Computer system ontology development pattern (2017), `http://ontologydesignpatterns.org/wiki/Submissions:Computer_System`, accessed 2020-12-01
16. Mohseni, M., et al.: Commercialize quantum technologies in five years. Nature **543**, 171–175 (2017)
17. Noy, N.F., Mcguinness, D.L.: Ontology development 101: A guide to creating your first ontology. Tech. rep., Stanford University (2001)
18. PlanQK: Planqk - platform and ecosystem for quantum-inspired artificial intelligence (2020), `https://planqk.de/en/`, accessed 2020-06-15
19. Preskill, J.: Quantum Computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
20. Publio, G.C., Esteves, D., Ławrynowicz, A., Panov, P., Soldatova, L., Soru, T., Vanschoren, J., Zafar, H.: Ml-schema: Exposing the semantics of machine learning with schemas and ontologies (2018)
21. Salm, M., Barzen, J., Breitenbücher, U., Leymann, F., Weder, B., Wild, K.: A Roadmap for Automating the Selection of Quantum Computers for Quantum Algorithms. arXiv preprint arXiv:2003.13409 (2020)
22. Schuld, M., Petruccione, F.: Supervised Learning with Quantum Computers. Quantum Science and Technology, Springer International Publishing (2018)
23. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: Openml: networked science in machine learning. SIGKDD Explorations **15**(2), 49–60 (2013)