1 QuantumShare: An Ontology to Share Information about Quantum Computing

This chapter describes and illustrates the ontological model that is the foundation of QuantumShare: an ontology to share information about quantum computing. In the development of this model, the tasks described in Section 3.2.2 were iteratively performed to satisfy the requirements stated in Section 5.3. Thereby, the results of these steps contribute to answering SRQ 6.

As emphasized throughout the methodology (Section 3.2), a modular ontology design is pursued, as it benefits maintainability, reusability, and the efficiency of reasoning (Stuckenschmidt & Klein, 2007). Figure 14 illustrates the modular approach used in the development of QuantumShare. As can be seen, QuantumShare integrates four modules: publication, quantum algorithm, implementation, and problem-execution. Additionally, two sub-modules, organization and parameter, are integrated into the implementation, problem-execution, and publication modules.

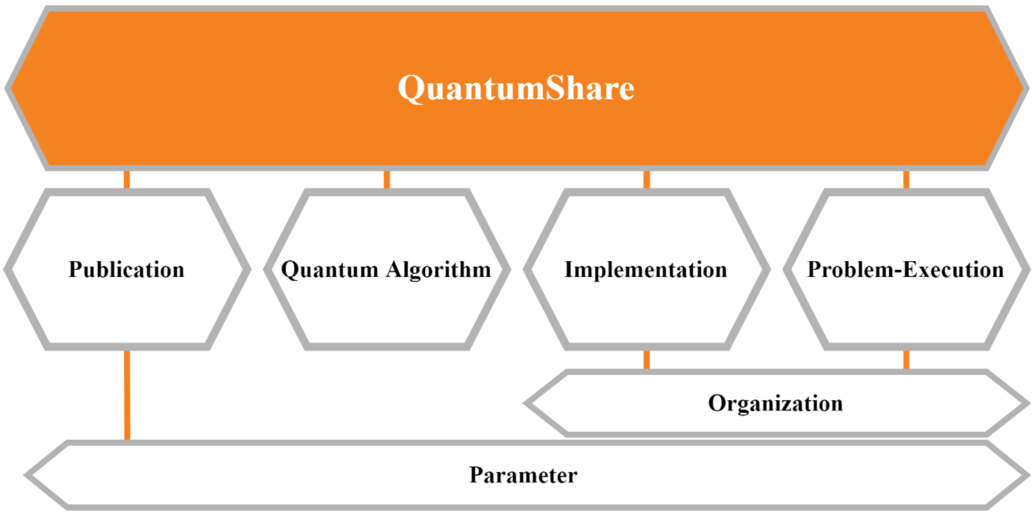

Figure 1, Modular structure of QuantumShare

Firstly, Section 6.1 presents the distinct modules used in the ontology design. Subsequently, Section 6.2 denotes how these modules were integrated into QuantumShare. Lastly, the evaluation results of QuantumShare are stated in Section 6.3.

1.1 QuantumShare Ontology Modules

All modules used in the development of QuantumShare are described throughout this chapter. As summarized in Table 4 (Section 3.2.2), a pattern language is used to document the modules' core structurally.

1.1.1 Parameter

Intent

The parameter module represents parameters, mainly used as an ontological model to describe properties and metadata, with a corresponding dimension and unit of measurement.

Competency Questions

-

Elements

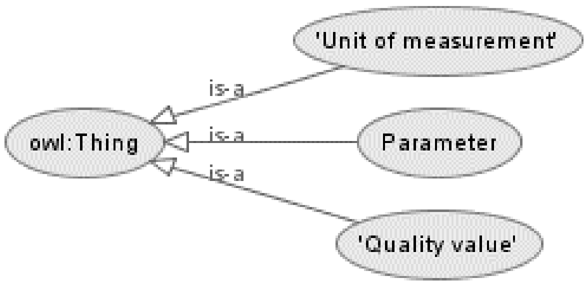

Figure 2, Class hierarchy of Parameter CP through the OWLViz plugin in Protégé

Reengineering Approach

The parameter module is constructed by integrating axioms of foundational ontology DUL with the Measurement Units Ontology (MUO) (Politecnico di Torino, 2020). By complementing DUL with MUO, physical properties and quantities can be semantically represented. Every parameter (i.e., *dul:Parameter*) parametrizes a certain quality (i.e., *muo:QualityValue*), which is a logical dimension of the parameter (e.g., length, weight). Furthermore, this dimension usually has units (i.e., *muo:UnitOfMeasurement*) in which it is measured (e.g., meters, kilograms). By representing a parameter's quality and unit of measurement, multiple observations of a particular parameter can be accurately compared (Bassiliades et al., 2018).

Scenarios

Parameters are specialized to resource properties in the implementation module (Section 6.1.6). For example, coherence time (i.e., resource property) parametrizes a duration (i.e., quality) measured in seconds (i.e., unit of measurement).

Furthermore, partial usage can be identified in the specialization to a software property. For example, version (i.e., software property) has data value 1.1.7 (i.e., literal).

Diagram

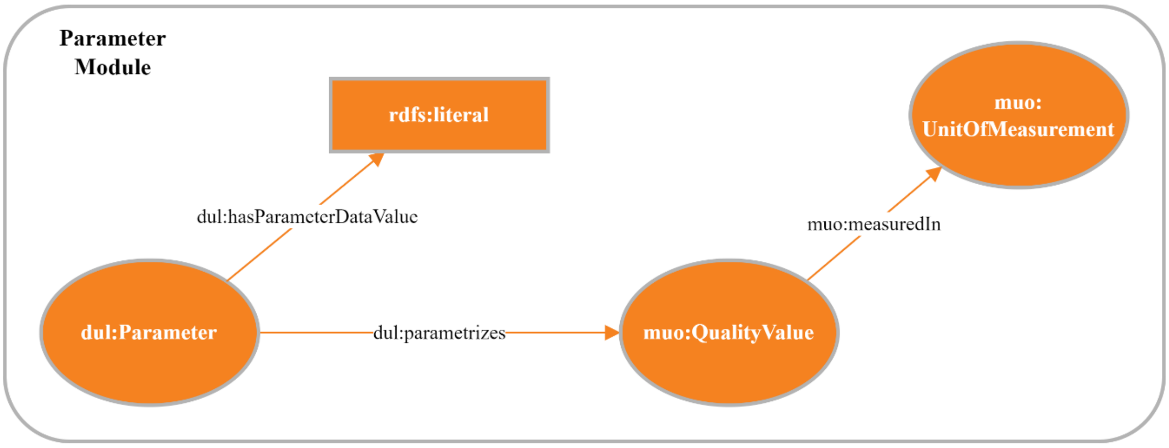

Figure 3, Parameter module

Building Block

https://github.com/julianmartens98/QuantumShare/blob/main/Modules/Parameter.owl

1.1.2 Organization

Intent

The organization module describes an organization, or part of an organization, and the classification and site that identify this organization.

Competency Questions

28, 29

Elements

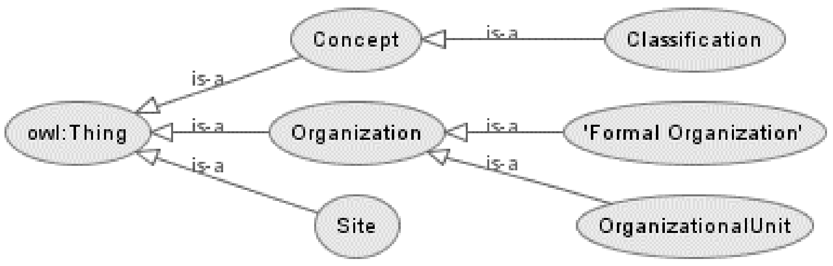

Figure 4, Class hierarchy of Organization CP through the OWLViz plugin in Protégé

Reengineering Approach

The organization module was created by cloning relevant axioms from the Organization Ontology of the World Wide Web Consortium (2014). The Organization Ontology describes organizational structures and is created to support data sharing of organizational information across various domains. Thereby, it matches the intent of this module. The following semantic relationships were derived from the Organization Ontology, in addition to the CQs:

- An organization is a collection of people organized together in a specific structure, having a shared purpose. However, this does not imply that organizations (i.e., *org:Organization*) are formally identified. For example, social networks are considered informal organizations, while corporations and universities are the opposite (i.e., *org:FormalOrganization*). Consequently, formal organizations are considered a subclass of organizations.
- Organizational units are considered a subclass of organizations to facilitate precision of the eventually stored instances. For example, a research department (i.e., *org:OrganizationalUnit*) is part of a university (i.e., *org:Organization*).

Scenarios

The research group of Matthias Troyer (i.e., organizational unit) is part of the ETH Zürich (i.e., formal organization). The ETH Zürich can be classified as a university of technology (i.e., classification). The research group of Matthias Troyer can be classified as well, as their research concerns computational physics (i.e., classification). The research group is located at the site “Building HIT, Wolfgang-Pauli-Str. 27, 8093 Zürich”.

Diagram

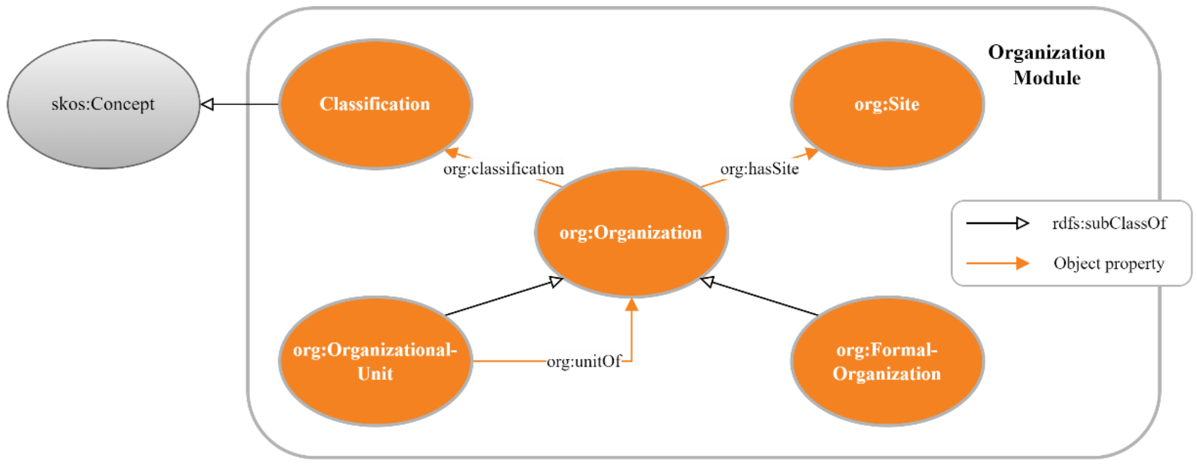

Figure 5, Organization module

Building Block

https://github.com/julianmartens98/QuantumShare/blob/main/Modules/Organization.owl

1.1.3 Publication

Intent

Representing a characterization of a (scientific) publication, used to reference algorithms, implementations, or executions.

Competency Questions

33

Elements

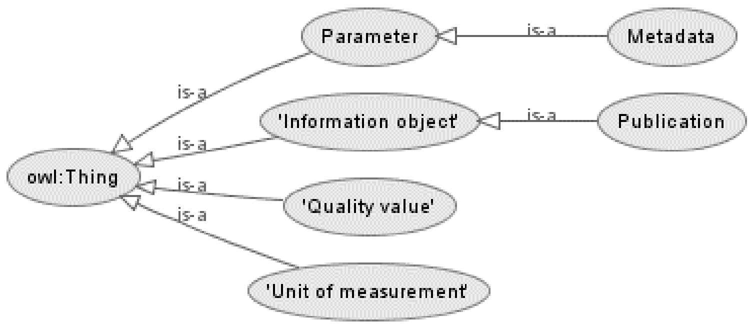

Figure 6, Class hierarchy of Publication CP through the OWLViz plugin in Protégé

Reengineering Approach

The publication module is developed by integrating and specializing foundational ontology DUL and the parameter module (Section 6.1.1). Firstly, a publication is considered a subclass of an information object (i.e., *dul:InformationObject*), as it is a piece of information (e.g., text). Furthermore, publications consist of metadata, which is a specialization of a parameter. Examples of metadata are authors, titles, and links (e.g., DOI or URI).

Scenarios

The authors (i.e., metadata) Dorit Aharonov, Itai Arad, Elad Eban, and Zeph Landau (i.e., parameter data value) wrote a publication titled (i.e., metadata) "Polynomial quantum algorithms for additive approximations of the Potts model and other points of the Tutte plane" (i.e., parameter data value). The publication can be found by its link (i.e., metadata) "http://arxiv.org/abs/quant-ph/0702008" (i.e., parameter data value).

Diagram

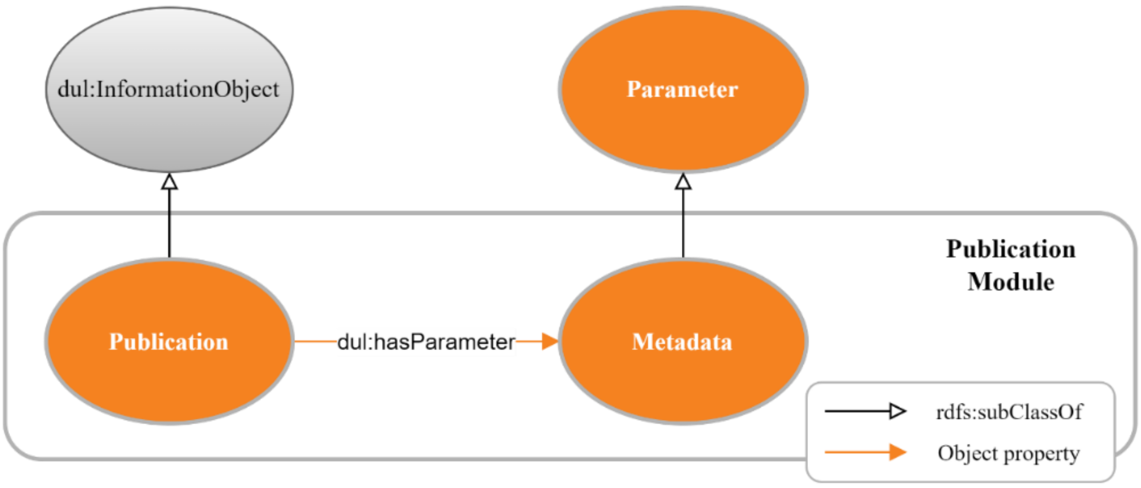

Figure 7, Publication Pattern

Building Block

https://github.com/julianmartens98/QuantumShare/blob/main/Modules/Publication.owl

1.1.4 Quantum Algorithm

Intent

Representing a quantum algorithm, which is an expression of a computational model, and its classifications to collections.

Competency Questions

CQ 6, 7, 8, 9

Elements

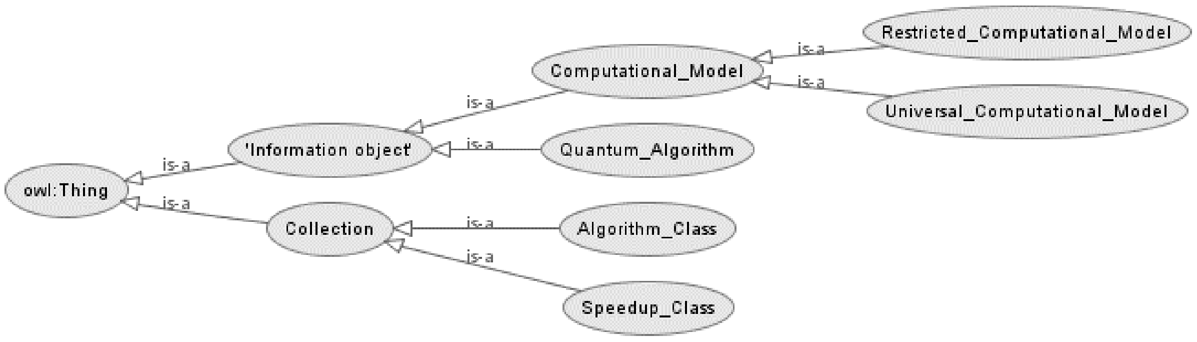

Figure 8, Class hierarchy of Quantum Algorithm CP through the OWLViz plugin in Protégé

Reengineering Approach

No existing CP exists, which could be reused to model the quantum algorithm knowledge. Hence, this module was created by partially cloning and specializing the foundational DUL ontology.

- Quantum algorithms can be distinguished into collections of algorithms with shared properties: speedup class (e.g., super-polynomial) and algorithm class (e.g., algebraic and number-theoretic algorithms). While speedup seems an instantaneous achievement, the degrees of speedup (e.g., polynomial or super-polynomial) are considered ranges and, consequently, collections. In this module, algorithms are related to these collections by the exemplary information object (i.e., quantum algorithm) – collection (i.e., algorithm class) relationship of foundational ontology DUL.

- The "comment" property of the RDF Schema is used to capture algorithm descriptions. RDF Schema provides fundamental axioms for RDF data (W3C, 2014b), and the comment property aims to provide descriptions of a resource.
- An algorithm is considered an expression of a computational model, as computational models contain mathematical rules that define how computation is performed (Vietz et al., 2021). An axiom of foundational ontology DUL is imported, letting information objects (i.e., quantum algorithms) express other information objects (i.e., computational models).

Scenarios derived from Jordan (2021)

- A factoring algorithm (i.e., quantum algorithm) is part of algebraic and number-theoretic algorithms (i.e., algorithm class). These algorithms solve the factorization problem in O(n^3), thereby it achieves super-polynomial speedup (i.e., speedup class).
- Constraint satisfaction algorithms (i.e., quantum algorithm) are part of optimization, numeric, and machine learning algorithms (i.e., algorithm class). Polynomial speedup is generally achieved (i.e., speedup class) over the fastest classical algorithm.

Diagram

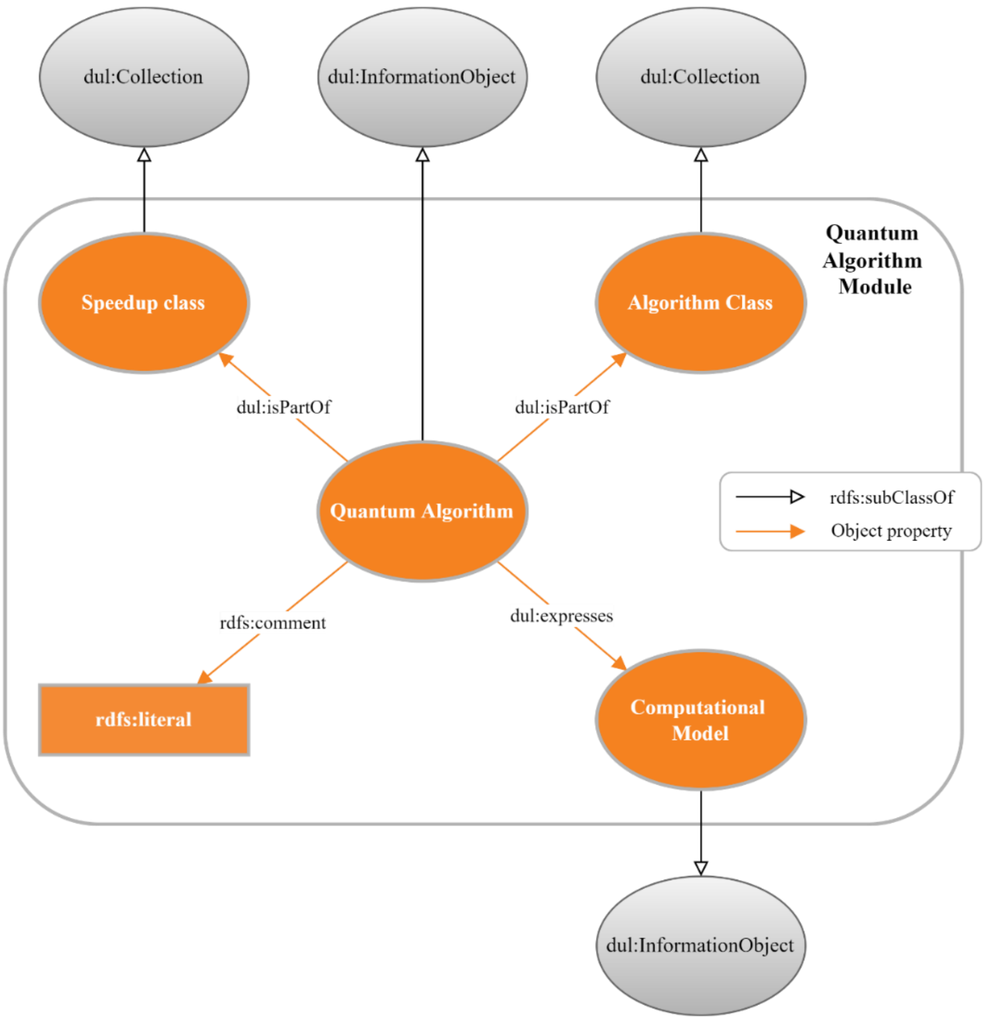

Figure 9, Quantum algorithm module

Building Block
https://github.com/julianmartens98/QuantumShare/blob/main/Modules/QuantumAlgorithm.owl

1.1.5 Problem-Execution

Intent
Representing the execution of an algorithm's implementation, with a particular performance, through which an organization's problem is solved.

Competency Questions
1, 2, 3, 4, 5, 25, 26, 27

Elements

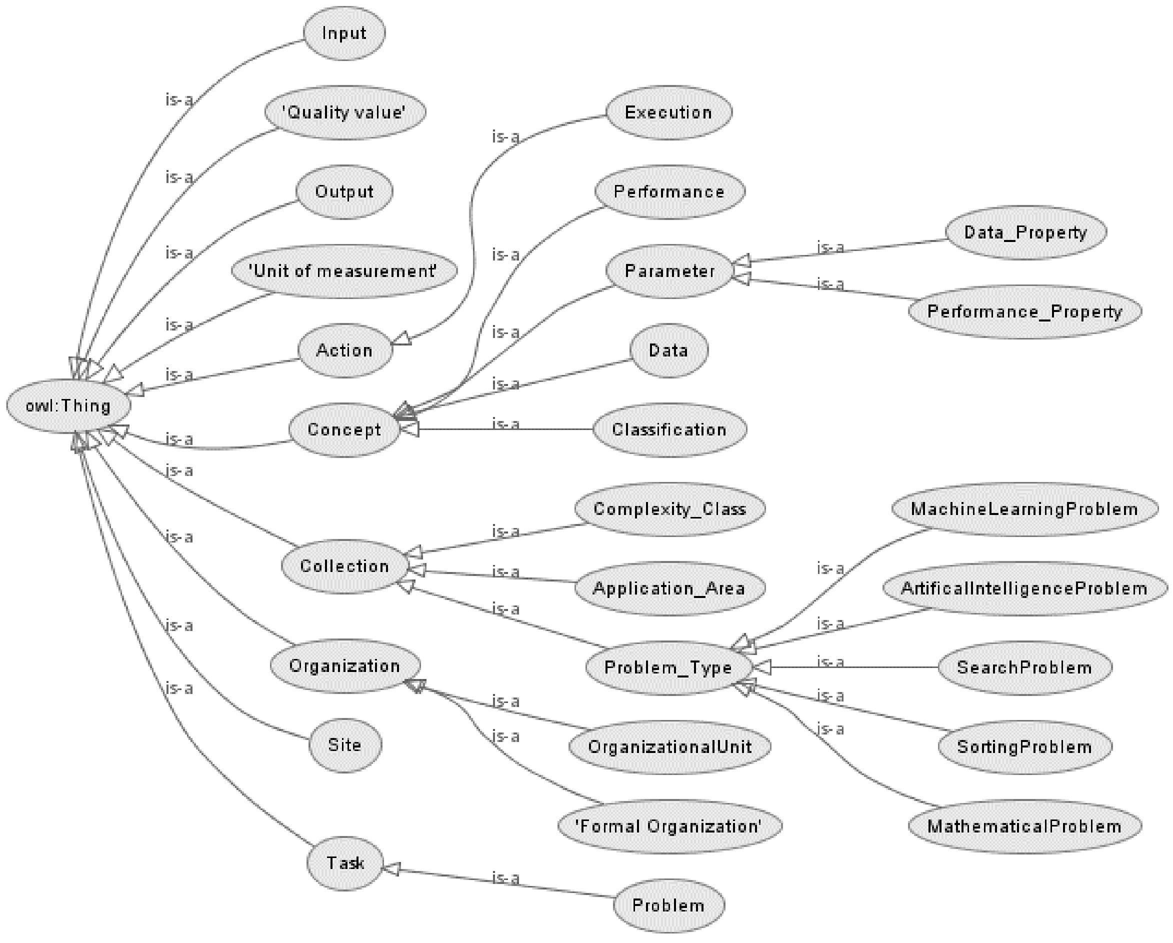

Figure 10, Class hierarchy of Problem-Execution CP through the OWLViz plugin in Protégé

Reengineering Approach
The core of the problem-execution module is derived by specializing the Task-Execution CP by Gangemi (n.d.). The Task-Execution CP, illustrated in Figure 45 in Appendix F, aims to portray actions through which tasks are executed, including a participating agent. The following specializations are performed to match this CP with our local domain:

- Execution is considered a specialization of action.
- The task, an event type that classifies an action (i.e., execution) to be executed, is specialized to a problem that needs to be solved (e.g., a traveling salesman problem).

- The agent participating in the action is specialized in an organization for which the organization module (Section 6.1.2) is imported.

Afterward, the problem-execution module was expanded with the following axioms:

- The “comment” property of the RDF Schema is used to capture problem descriptions. RDF Schema provides fundamental axioms for RDF data (W3C, 2014b), and the comment property aims to provide descriptions of a resource.
- Problems can be distinguished into collections of problems with shared properties, such as the complexity class, problem type, and application area. These collections are captured through the collection class of foundational ontology DUL.

Next, the execution is specified further by partially integrating the ML-Schema CP by Ławrynowicz et al. (2016). This CP, which can be seen in Figure 46 in Appendix F, intends to “model … executions, together with parameters of implementations, settings of the parameters for the execution, and inputs the execution consumes (e.g., data) and outputs the execution produces (e.g., models, reports)”. Ławrynowicz et al. (2016) created a relatively generic vocabulary, which eased the integration of their CP with the problem-execution module. As can be seen in Figure 24, the input and output classes from the ML-Schema CP, and their object properties, are cloned in the problem-execution module. In addition, the following expansions are made:

- Firstly, input defines the data and parameters involved. Secondly, output defines performance. The exemplary description-concept axiom of the foundational ontology DUL is used, where description refers to input and output, and concept to data and performance.
- Specifying data and performance is facilitated by their relationship with data and performance properties. These properties are subclasses of the parameter class imported from the parameter module (Section 6.1.1).

Scenarios derived from (Harwood et al., 2021)

ExxonMobil and IBM (i.e., organizations) collaboratively explored quantum solutions to routing problems. They aimed to solve a vehicle routing problem with time windows (VRPTW) (i.e., problem) in the context of maritime shipping (i.e., application area). Vehicle routing problems are considered NP-hard (i.e., complexity class) and belong to combinatorial optimization problems (i.e., problem class). ExxonMobil and IBM approached the problem as QUBO (i.e., quadratic unconstrained binary optimization) with an ADMM solver (i.e., alternating direction method of multipliers) (i.e., execution). Mathematical formulations (i.e., input) were composed of parameters for the routes traveled, feasible movements between customers and ports, and the order in which locations are visited (i.e., parameter). Furthermore, explicit parameters were set, such as the maximum number of function evaluations (i.e., parameter) with a value of 1000. The execution enabled numerical experiments (i.e., output) that defined multiple evaluation metrics (i.e., performance). These metrics included the probabilities for success and feasible solutions, the number of iterations, and the number of qubits (i.e., performance properties), with a value of 16.

Diagram

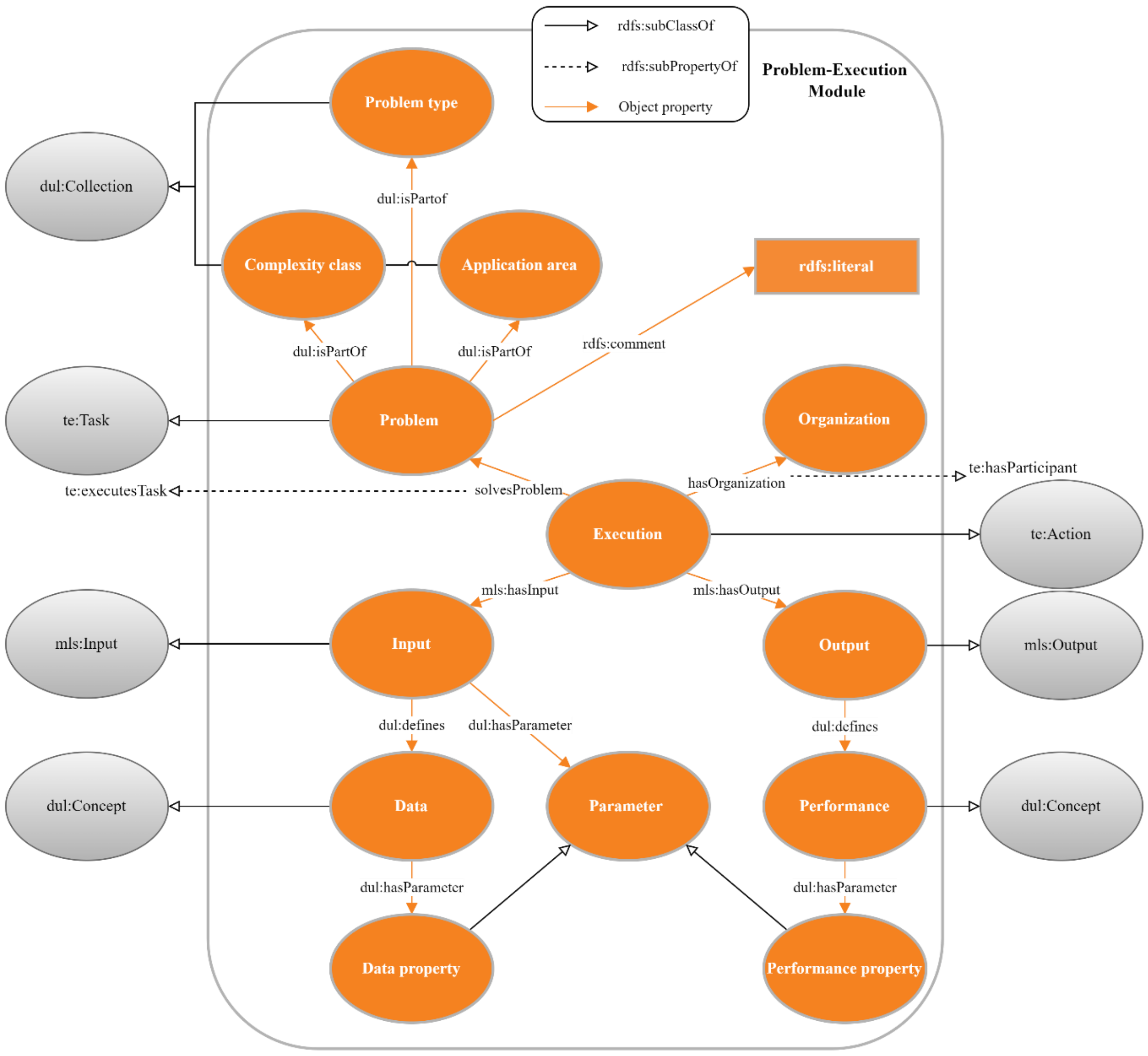

Figure 11, Quantum problem-Execution ontology module

Building Block

https://github.com/julianmartens98/QuantumShare/blob/main/Modules/ProblemExecution.owl

1.1.6 Quantum Implementation

Intent

The quantum implementation module represents an orchestration of software and hardware that is able to implement a quantum algorithm.

Competency Questions

12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23

Elements

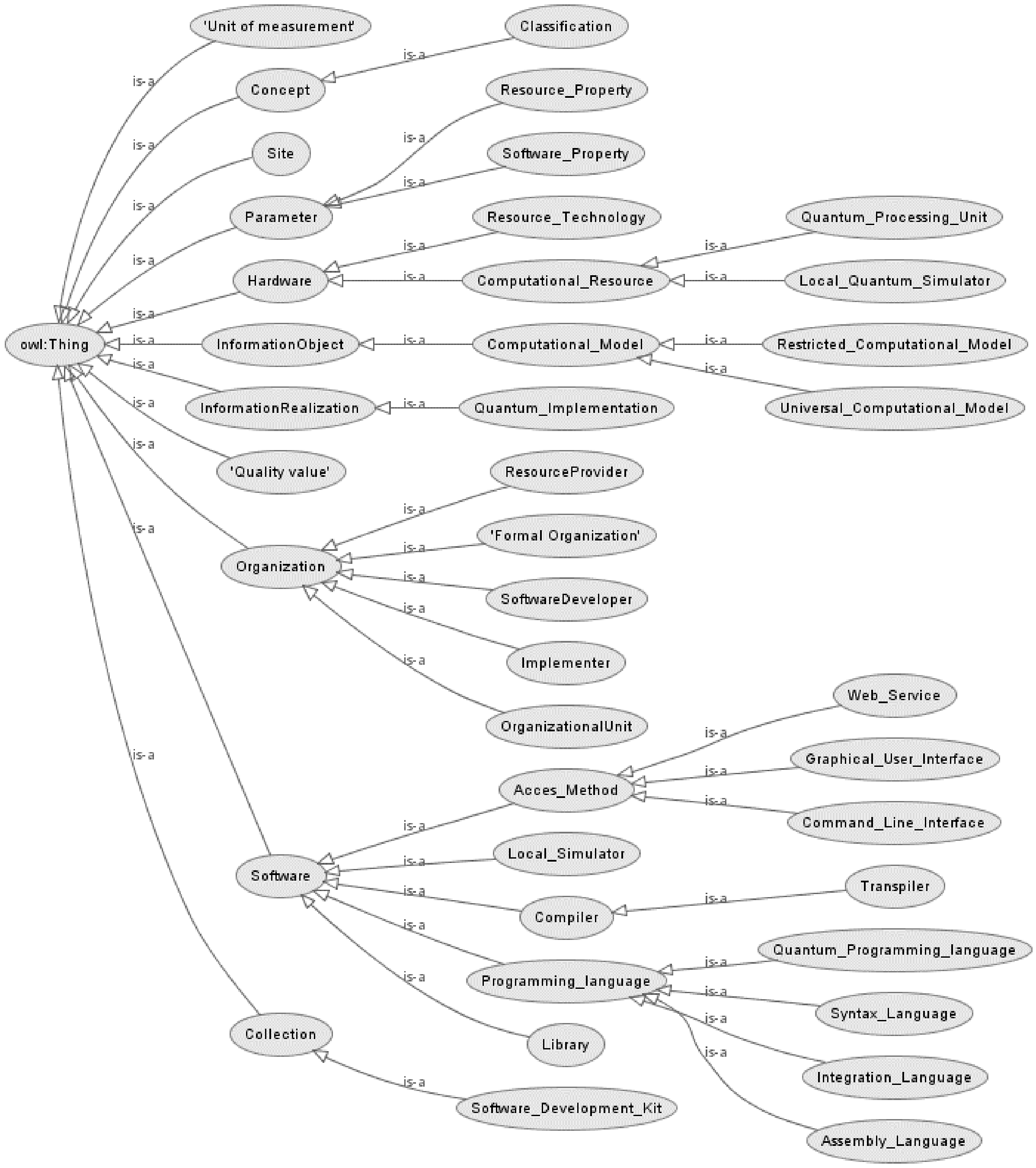

Figure 12, Class hierarchy of Quantum Implementation CP through the OWLViz plugin in Protégé

Reengineering Approach

The quantum implementation module is created by complementing the Computer-System CP by Mitzias et al. (n.d.) with foundational ontology DUL. Most axioms in the module are deducted from the Computer-System CP (as seen in Figure 47 in Appendix F), which intends to model computer systems by defining relationships between software and hardware.

Firstly, a quantum implementation uses software and hardware, and the latter specializes in computational resources in the quantum implementation module. Then, the following additional axioms are defined:

- Computational resources (i.e., *cs:Hardware*) are either quantum processing units (QPU) or local quantum simulators. Furthermore, these resources require a technology to operate, which is also considered hardware.
- Access methods (e.g., web services, graphical user interfaces, or command-line interfaces), libraries, local simulators, programming languages (e.g., assembly, quantum programming, integration, and syntax), and compilers are considered subclasses of software (*cs:Software*). Furthermore, a transpiler is a type of compiler, making it a subclass of the latter.
- Software is often hardware-dependent (Vietz et al., 2021). Hence, the *cs:isCompatibleWith* property intends to capture this dependency.

Furthermore, the quantum implementation module differs from the Computer-System CP in its expansion with additional axioms:

- A computational resource serves particular computational models (i.e., *dul:InformationObject*). Consequently, a computational model should be compatible with a computational resource.
- The parameter module (Section 6.1.1) is consulted to capture the properties of software and computational resources. Accordingly, software and resource properties are subclasses of a parameter.
- Compilers, libraries, local simulators, and access methods are often contained in a software development kit (SDK). Therefore, an SDK is modeled as a specialization to DUL's collection class. However, these software types are also used as standalone technology.
- Three types of organizations are involved in the implementation of a quantum algorithm. Computational resources involve a resource provider, software involves a software developer, and quantum implementations involve an organization that implements and organizes all the software and hardware. The organization module (Section 6.1.2) is consulted to serve as the superclass of these actors.

Scenarios retrieved from Larose (2019)

A quantum algorithm can be implemented through Project Q (i.e., quantum implementation), a framework developed by the research group of Matthias Troyer (i.e., implementing organization). The ProjectQ software properties are version 0.3.6, license Apache-2.0, and it operates on Mac, Windows, or Linux systems. It can be implemented using its ProjectQ quantum programming language, hosted on Python (i.e., syntax language). Furthermore, ProjectQ can connect to IBM's (i.e., resource provider) backend for the computational resource IBMQX5 (i.e., quantum processing unit). If so, OpenQASM (i.e., assembly language) will also be used. One can also use ProjectQ's C++ simulator (i.e., local quantum simulator), which has 28 qubits and a gate set of 20 (i.e., resource properties).

Diagram

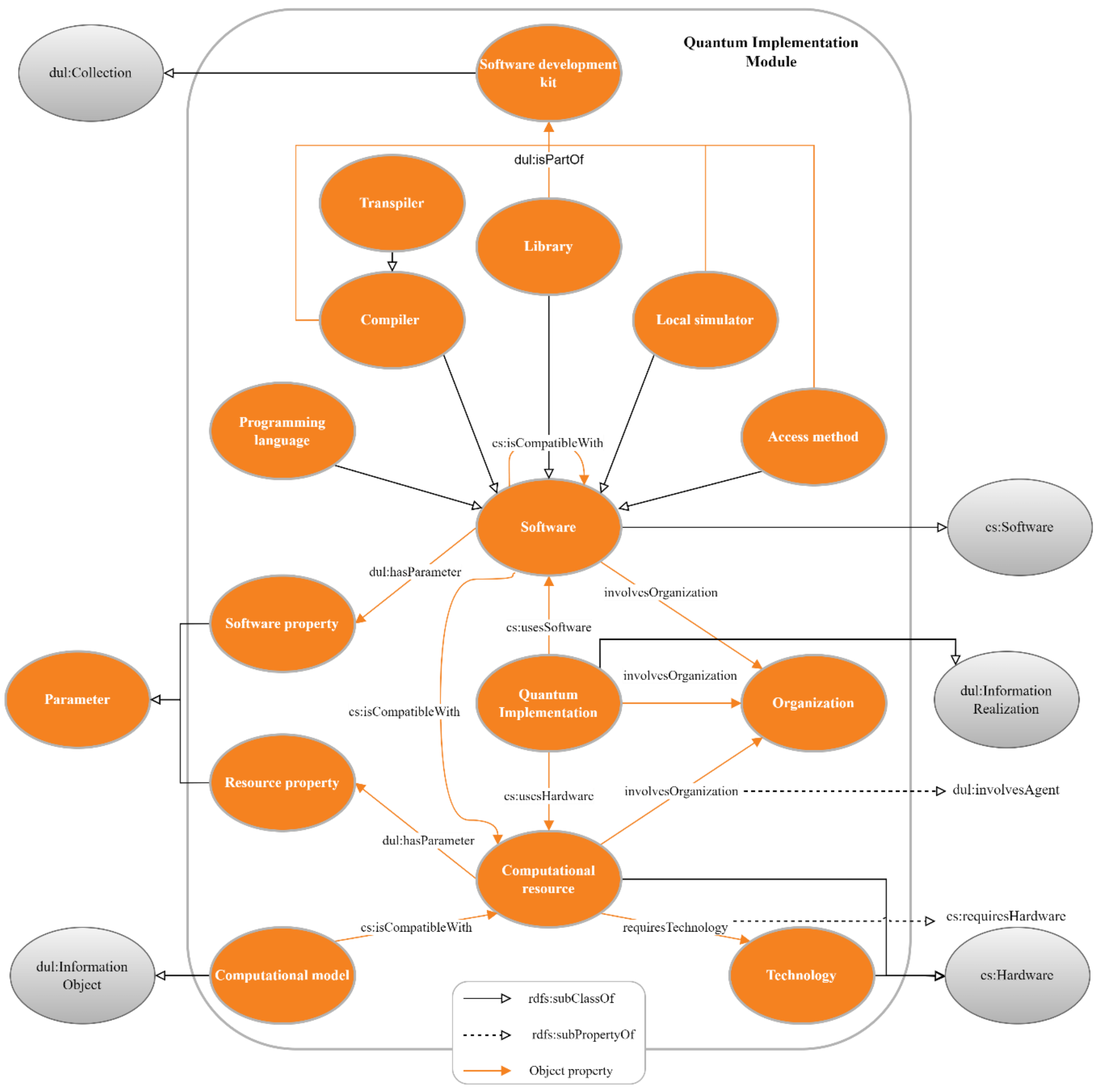

Figure 13, Quantum implementation module

Building Block

https://github.com/julianmartens98/QuantumShare/blob/main/Modules/QuantumImplementation.owl

1.2 Module Integration: QuantumShare

Intent

To describe quantum computing use cases by providing the algorithm, implementation, and problem-solving execution of the algorithm.

Additional Competency Questions

10, 11, 24, 30, 31, 32

Elements

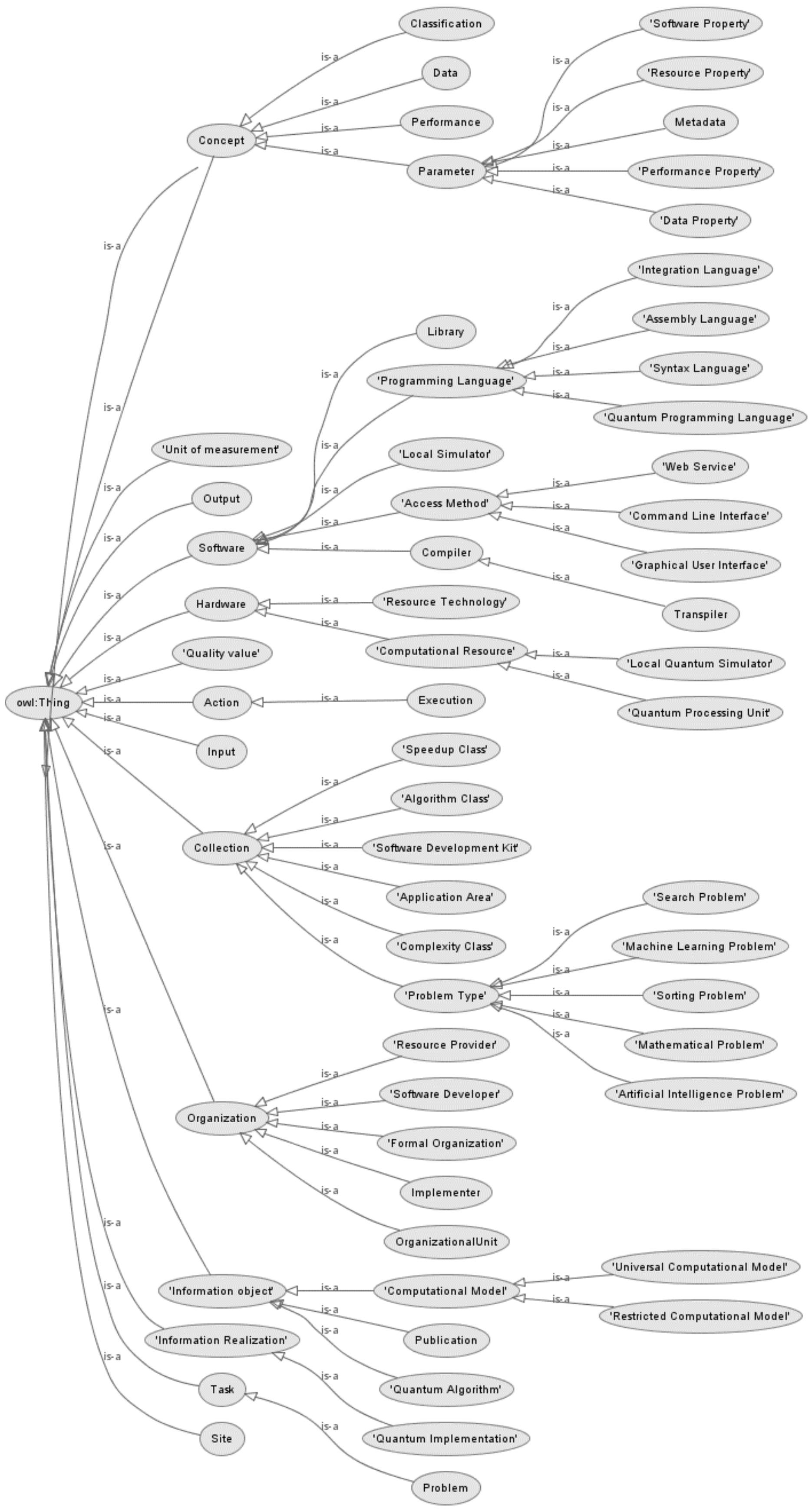

Figure 14, Class hierarchy of QuantumShare through the OWLViz plugin in Protégé

Reengineering Approach

The QuantumShare ontology is established by integrating the ontology modules described throughout Section 6.1, based on the methodology defined in Task 10 in Section 3.2.2. Integration is carried out by creating relationships across the four major classes of each ontology module (i.e., publication, quantum algorithm, quantum implementation, and problem-execution). These relationships serve as an upper-level ontological model, as seen in Figure 28. Correspondingly, foundational ontology DUL, which is considered an upper-level ontology, is consulted to define the following relationships:

- Publications provide the information source which references the knowledge captured in the quantum algorithm, implementation, and problem-execution module. The object property that relates the publication to the central classes of these modules (i.e., *dul:isAbout*) relates information objects to other entities.
- Quantum implementation (i.e., *dul:InformationRealization*) and execution (i.e., *dul:Action*) are considered actions that enable the realization of a quantum algorithm. Therefore, the object property *dul:realizes* is dedicated to these types of relationships.
- Lastly, execution is considered the context-dependent part of the quantum implementation, providing the problem, organization, and instantaneous performance measures to a quantum implementation. DUL allows the usage of this relationship between all entities.

Scenarios

The publication of Harwood et al. (2021) titled "Formulating and Solving Routing Problems on Quantum Computers" (i.e., captured in publication module) provides research about the use of variational algorithms, such as the variational quantum eigensolver (VQE) and the quantum approximate optimization algorithm (QAOA) (i.e., captured in quantum algorithm module). Harwood et al. (2021 realized the implementation of these algorithms by accessing IBM's quantum simulators, accessed with SDK Qiskit (i.e., captured in quantum implementation module). Part of this implementation aimed to optimize specific mathematical formulations in maritime shipping (i.e., problem-execution module).

Diagram

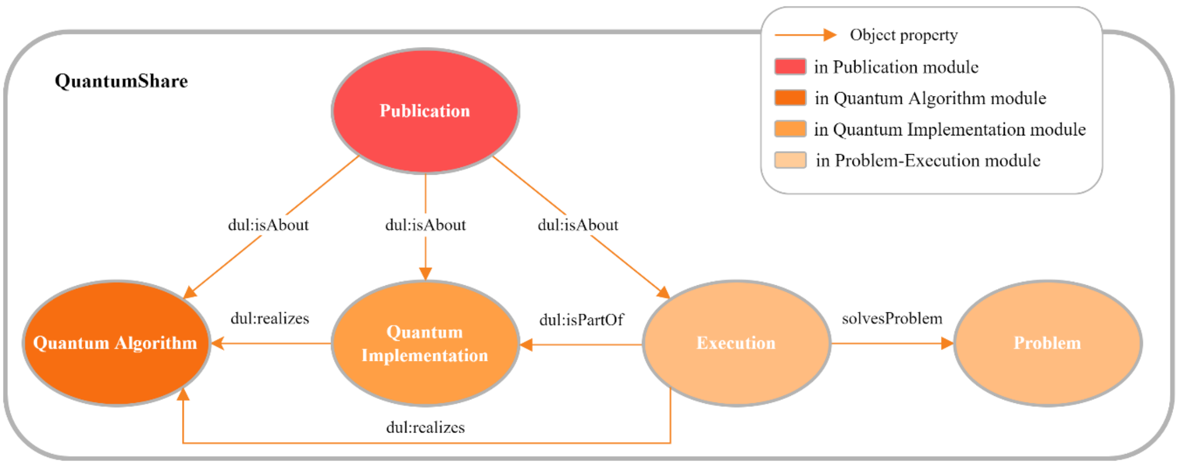

Figure 15, QuantumShare, integration of modules

Building Block

https://github.com/julianmartens98/QuantumShare/tree/main/QuantumShare%20schema

1.3 Evaluation of Structure, Functionality, and Usability

Throughout the development of the ontology modules and QuantumShare, the quality of the ontology (modules) was continuously evaluated, which resulted in iterations in the development process. This chapter states the concluding evaluation results.

Ontology evaluation tool OOPS (i.e., Ontology Pitfall Scanner) is used to provide a final evaluation of QuantumShare. OOPS evaluates 41 criteria related to structure, functionality, and usability. Each criterion is weighted with an importance level (i.e., critical, important, and minor). Where critical criteria are crucial to correct, minor criteria do not comprise the ontology and do not result in problems (Poveda-Villalón et al., 2014).

The final evaluation of QuantumShare identified four issues in the ontology (as seen in Table 15), of which criterion 41 is considered out of scope for this research. The remaining issues all relate to the ontology's usability. Firstly, upper-level object properties may miss a domain or range due to partially cloning their complete axiom. However, the four denoted cases concern object properties of DUL that do not directly contribute to the QuantumShare ontology. Furthermore, nine inverse relationships were not defined, referring to reused CPs. For example, the ML-Schema CP and Computer-System CP do not define inverse relationships, as seen in Figure 46 and Figure 47 in Appendix F. Lastly, QuantumShare follows a different naming convention than the reused CPs, which results in a minor pitfall. More specifically, QuantumShare uses the "snake case" naming convention (e.g., Quantum_Algorithm), while DUL uses the "camel case" (e.g., QuantumAlgorithm).

Table 1, Evaluation results from Ontology Pitfall Scanner (*: applies to entire ontology)

Criterion	Description	# cases	Importance level
11	Missing domain or range in properties	4	Important
13	Inverse relationships not explicitly declared	9	Minor
22	Using different naming conventions in the ontology	*	Minor
41	No license declared	*	Important