Quantum Factory Method: A Software Engineering Approach to Deal with Incompatibilities in Quantum Libraries

Samuel Magaz-Romero[0000−0001−6438−5569], Eduardo Mosqueira-Rey[0000−0002−4894−1067], Diego Alvarez-Estevez[0000−0001−5790−0577], and Vicente Moret-Bonillo[0000−0002−9435−3151]

Universidade da Coruña, CITIC, Campus de Elviña, 15071 A Coruña, España
{s.magazr,eduardo,diego.alvareze,vicente.moret}@udc.es

Abstract. The current context of Quantum Computing and its available technologies present an extensive variety of tools and lack of methodologies, leading to incompatibilities across platforms, which end up as inconsistencies in the developed solutions. We propose a design called Quantum Factory Method, based on software engineering and design patterns, to solve these issues by integrating different quantum platforms in the same development. We provide example implementations whose results prove the suitability of the design in different cases, and conclude on how this approach can be expanded for future work.

Keywords: Quantum Computing · Software Engineering · Design Patterns · Rule-Based Systems · Uncertainty

1 Introduction

The exponential growth that has been experienced in recent years of the interest towards the field of Quantum Computing (QC) is undeniable [4]. Despite its theoretical basis being around since the 80s [8], the manufacturing of quantum computers and their availability has put this field in the eyes of many.

Some of the most relevant groups that are leading this new field are IBM [12] and Google [5], both of which offer different commercial solutions for QC. Other important competitors are Amazon [1] or Atos [2]. These are some names in the industry, but more are involved, showing the size of the QC environment so far.

From a Software Engineering (SE) perspective, this varying environment can end up being detrimental. The lack of standardisation makes each group approach their solution differently, and while programming libraries in Python are the most popular solution, these present differences on some core concepts.

Libraries are designed to work on the platform (programming tools and execution stack) of their company. There are some common factors among them, but they still present different philosophies and forms for users to interact with them. This complicates the introduction into the QC world, as well as the work for developers, dealing with the differences between several platforms.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6

All these factors make the comparison between options more troublesome; for example, when selecting the appropriate platform for a project. Each platform presents advantages and inconveniences regarding development, maintainability and usage, so the decision should be made taking these into account.

Therefore, the objectives that our approach must accomplish are: (i) to allow for a single problem definition to be used in different platforms, without addressing their philosophies for each case, (ii) to obtain standard outputs across platforms, facilitating their comparison, and (iii) to have the possibility to include new platforms in the future as they are made available to the general public.

2 State of the art

We present a brief excerpt of QC's state of the art, specifically on the current situation of Quantum Software Engineering and the OpenQASM standard.

2.1 Quantum Software Engineering

While there is a lack of Quantum Software Engineering methodologies, they exist, but there is not a consensus like on classical methodologies. It seems logical due to Quantum Computing being on its early stages, yet it could benefit from tools and procedures in order to keep expanding its horizons [15,17,10].

There is interest in the subject, but no proposal for a methodology of Quantum Software Engineering has firmly established itself as the proper approach, due to: (1) Quantum Computing is not as software-engineer oriented as other branches of computing, drawing along users not familiarised with software engineering methodologies, and (2) a general lack of interest in methodology innovation, leading to not focusing on methodologies as much it would be needed.

In summary, there is no procedure or methodology (yet) to rely on when developing software using Quantum Computing.

2.2 OpenQASM: a not-so-standard standard

Open Quantum Assembly Language (OpenQASM) is an imperative programming language designed for near-term QC algorithms and applications. Programs are described using the measurement-based quantum circuit model with support for classical feed-forward flow control based on measurement outcomes [7].

It was proposed by the IBM Quantum Computing group as an imperative programming language for quantum circuits. Since OpenQASM 2 was introduced, it has become a *de facto* standard in the field.

Nowadays, OpenQASM 3 includes better support for the next phase of quantum system development, as well as to incorporate some of the best ideas that have arisen in other circuit description languages [6].

However, OpenQASM is not a standard *per se*, since each quantum programming library implements it differently. Table 1 illustrates this situation [12,5,1,3,2]. OpenQASM is a suitable option for cases where only basic features are used, but it is not enough for the variety of problems we want to address here.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6

Table 1. OpenQASM's features supported in each platform
S = Supported P = Partially supp. N = Not supp. X = Unavailable in version 2.0

Platform		Qiskit	Cirq	Braket	Pennylane	myQLM
Version	2.0	S	S	S	S	S
	3.0	S	N	S	S	N
Main types	qubit	P	S	S	P	S
	bit	S	S	S	S	S
	bool	S	X	S	S	X
Secondary types	uint	N	X	S	N	X
	float	N	P	S	N	S
	complex	N	X	S	N	X
Gates	Basis	S	S	S	S	S
	Custom	N	S	S	N	S
	Control (on non-basis gates)	N	X	S	N	X
	Barrier	S	N	S	S	S
Flow	if	S	N	S	S	S
	else	S	X	S	S	X
	else if	N	X	S	N	X
	for	P	X	S	P	X
Subroutines		N	X	S	N	X

3 Proposal

Our proposal is based on the use of design patterns, which eases the elaboration of an object-oriented design and its understanding for other developers.

3.1 Design patterns

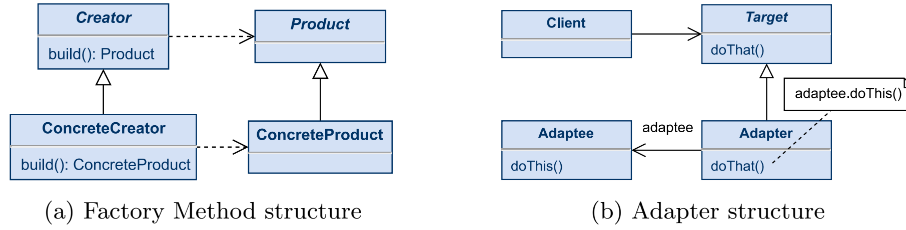
In SE, a design pattern [16] is a general repeatable solution to a commonly occurring problem in software design. It is a description or template for how to solve a problem that can be used in many different situations.

Design patterns provide a common vocabulary that eases the sharing and understanding of concepts, preventing subtle issues that can cause major problems, improving code readability for coders and architects familiar with them. The design patterns commonly known in the general literature are the ones presented on [9]; the ones our solution needs are Factory Method and Adapter (Figure 1).

Factory Method is a creational pattern to deal with the problem of creating objects without having to specify the exact class. This is done by creating objects by calling a method that encapsulates a constructor. The `Creator` class defines a method to build `Product` objects, but lets the subclasses like `ConcreteCreator` decide which class to instantiate, in this case `ConcreteProduct`.

Adapter is a structural pattern that converts the interface of a class into another, creating an intermediary abstraction that translates the old component to the new system. The `Client` interacts with the `Target` class, from whom expects the interface `doThat()`. The `Adaptee` class offers the interface `doThis()`, which is converted by the `Adapter` class into the expected `doThat()`.

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6



(a) Factory Method structure (b) Adapter structure

Fig. 1. Design patterns' structures

3.2 Application

We can now introduce the final design of our proposal, illustrated in Figure 2.

The `Platform` class (representing a quantum platform) acts as the `Creator` from the Factory Method design pattern, and the `Circuit` class (representing a quantum circuit) acts as the `Product`, including a subclass of each per quantum platform. Regarding the Adapter design pattern, the `Circuit` class acts as the `Target`, and each subclass is the `Adapter` of their quantum library.

The `build` method can receive an input to be translated into the quantum circuit. Therefore, with a single definition, the same circuit can be built in different platforms and run on different quantum computers.

The `execute` method can return an object with the results, easing the comparison of platforms, for example for benchmarking studios.

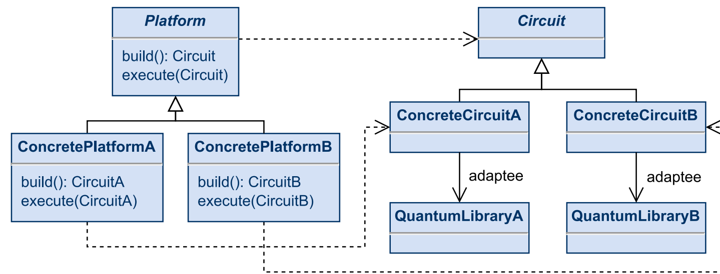


Fig. 2. Quantum Factory Method structure

4 Examples

Two examples were developed, one using OpenQASM to build simple circuits and other building Quantum Rule-Based Systems (QRBS) [14] with inferential circuits. We use Qiskit and Cirq as platforms; code for the implementation is in [13].

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6

4.1 Building simple circuits

We define the classes required: one concrete `Platform` class and one concrete `Circuit` class per platform, and a `Result` class to store the values.

In this case we use the OpenQASM standard to define simple quantum circuits. This definition is represented on a string used by the platforms indistinctly. The `Circuit` classes encapsulate the quantum implementations for each library, which are obtained in the `build` method and used in the `execute` method. The `Result` class stores the values from the measurements of each qubit after the execution of the quantum circuit, as real numbers in the range $[0, 1]$.

With these elements defined, we obtain the design illustrated in Figure 3.

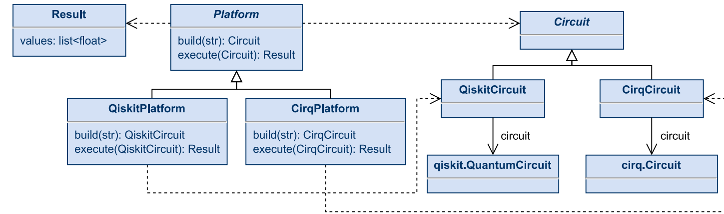


Fig. 3. OpenQASM example's design

4.2 Building Quantum Rule-Based Systems

Rule-Based Systems (RBS) are systems commonly used in Artificial Intelligence that encode and represent the knowledge of an expert in rules [11]. These rules are composed by a precedent and a consequent. Both are logical statements, known as facts, that can be evaluated as *true* of *false*.

Precedents can be formed combining several facts with logical operators like AND, OR, and NOT. Consequents can act as the precedent for other rules, allowing for the chaining of several rules. For example, the following rules are chained one after another and conform an inferential circuit:

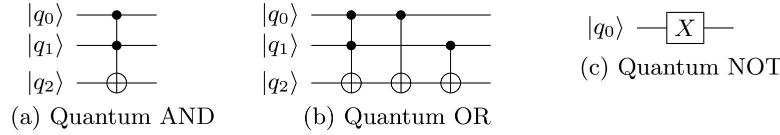
$$A \text{ AND } B \Rightarrow C \qquad C \text{ OR } D \Rightarrow E \qquad E \text{ AND } (F \text{ OR } G) \Rightarrow I$$

However, facts can be between *true* or *false* states. This idea is called uncertainty, and can be implemented in RBS. Using QC to represent uncertainty gives birth to QRBS. Overall, the elements that conform a QRBS are:

- **Facts**: single qubits, with state $|0\rangle$ being false and state $|1\rangle$ true.
- **Operators**: quantum operators, as illustrated in Figure 4.
- **Uncertainty**: a parameterized operator (Eq. 1) that puts a qubit in superposition, mapping its uncertainty degree in $[0, 1]$ to an angle in $[0, \pi/2]$.

$$M(\delta) = \begin{bmatrix} cos(\delta) & sin(\delta) \\ sin(\delta) & -cos(\delta) \end{bmatrix} \tag{1}$$

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6



(a) Quantum AND (b) Quantum OR (c) Quantum NOT

Fig. 4. Quantum logical operators

We can now apply the proposed design to build QRBS: we need to define one concrete `Platform` and `Circuit` per platform we want to implement, and a `Result` class to store the values obtained after measuring. In this case, the `Platform` class provides a `build` method that receives an inferential circuit as the input, and an `execute` method that receives the `Circuit` object and returns a `Result` object with the measured values. To represent classical inferential circuits we use the Composite design pattern, used to model tree structures that represent part-whole hierarchies. For the building process we incorporate the Visitor design pattern, where quantum platforms visit the elements of an inferential circuit to build its quantum circuit. The `Result` class stores the values obtained after measuring in a key-value dictionary, where the key is the tag of the element and the value is the measurement obtained. With these elements defined, we obtain the design shown in Figure 5.

4.3 Experiments and results

At this point, we can experiment with the examples as follows: (1) define a proper problem (an OpenQASM string or an inferential circuit), (2) call the `build` methods to obtain the `Circuit` objects, (3) call the `execute` methods to obtain the `Result` objects, and (4) analyse the obtained values to conclude.

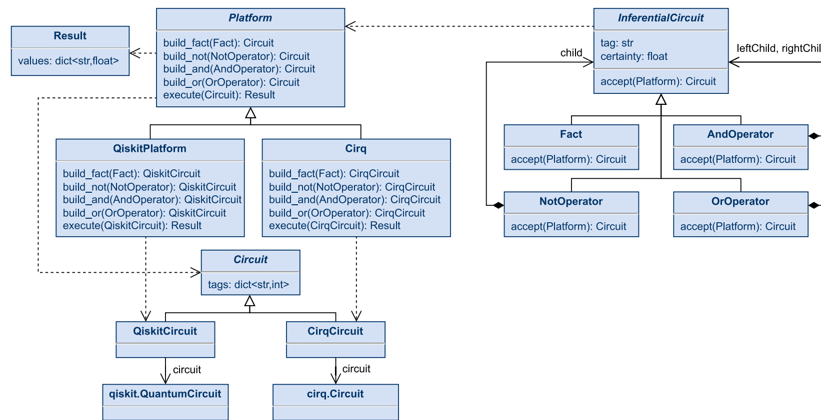


Fig. 5. QRBS example's design

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6

Table 2. Results for OpenQASM example with the Bell State circuit

	State probability	
Platform	$\|00\rangle$	$\|11\rangle$
Qiskit	0.511	0.489
Cirq	0.491	0.509

Table 3. Results for QRBS example with the proposed circuit

	Element								
Platform	A	B	C	D	E	F	G	H	I
Qiskit	1.000	0.331	0.286	0.373	0.416	0.214	0.950	0.838	0.344
Cirq	1.000	0.342	0.295	0.381	0.423	0.232	0.944	0.823	0.341

We define a proper problem for each example (a Bell state quantum circuit for OpenQASM and an inferential circuit with the rules of section 4.2 for QRBS), and execute them with the default parameters of each platform (local simulators, 1024 shots), obtaining the results shown in Tables 2 and 3. The values are similar but not identical, in part due to QC's probabilistic nature, yet they show the relevance of comparing the results obtained by different quantum platforms.

5 Discussion and conclusions

The design obtained, named Quantum Factory Method, is a robust and solid product, as a result of using design patterns. Its simplicity resides on covering the needs and modelling through the Factory Method and Adapter patterns.

The QRBS example illustrates the potential of our approach, as a single definition of an inferential circuit is built in each platform, without having to rebuild for each experiment. The abstraction allows to design the quantum inferential circuits without dealing with the specifics of the platforms. In this case, we could vary the certainty values of the elements without having to modify their quantum implementation, speeding up experiments carried out.

This design focuses on the higher levels of QC, facilitating the work for end users and delegating the specifics to software experts. Reviewing the initial objectives, we can observe: (i) through the abstractions, a single definition is passed onto the corresponding classes to carry on with the tasks, (ii) the output for the `execute` method can be standardised, and (iii) new platforms can be included into the design without modifying the ones contemplated.

The flexibility provided by the input to build the quantum circuits enables the representation of complex data structures with ease. Its range is illustrated with the examples of section 4, going from simple quantum circuits to complex cases like quantum algorithms, where more information is needed. With this design, one definition is enough for several quantum platforms. This workflow eases development and maintenance, which is key to develop large software products.

Regarding future work, we intend to keep looking for this kind of synergies. We believe that Software Engineering will be interesting for better develop Quan-

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6

tum Computing algorithms. We hope the ideas presented in this paper conform a step towards that direction.

Acknowledgements. This work has been supported by the European Union's Horizon 2020 under project NEASQC (grant agreement No 951821) and by the Xunta de Galicia (grant ED431C 2022/44) with the European Union ERDF funds and Centro de Investigación de Galicia "CITIC", funded by Xunta de Galicia and the European Union (European Regional Development Fund-Galicia 2014-2020 Program, grant ED431G 2019/01). DAE received funding from the project ED431H 2020/10 of Xunta de Galicia.

References

1. Amazon Web Services: Amazon Braket. https://aws.amazon.com/braket/, accessed: 23-11-2022
2. Atos: myQLM. https://myqlm.github.io/, accessed: 23-11-2022
3. Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Blank, C., McKiernan, K., et al.: Pennylane: Automatic differentiation of hybrid quantum-classical computations (2018). https://doi.org/10.48550/arxiv.1811.04968, accessed: 23-11-2022
4. Biondi, M., Heid, A., Henke, N., Mohr, N., Pautasso, L., Ostojic, I., Wester, L., Zemme, R.: Quantum computing: An emerging ecosystem and industry use cases. McKinsey & Company (2021)
5. Cirq Developers: Cirq (Apr 2022). https://doi.org/10.5281/zenodo.6599601
6. Cross, A., Javadi-Abhari, A., Alexander, T., Beaudrap, N.D.: OpenQASM 3: A broader and deeper quantum assembly language. ACM Transactions on Quantum Computing **3**(3), 1–50 (sep 2022). https://doi.org/10.1145/3505636
7. Cross, A.W., Bishop, L.S., Smolin, J.A., Gambetta, J.M.: Open quantum assembly language (2017). https://doi.org/10.48550/arxiv.1707.03429
8. Feynman, R.P.: Quantum mechanical computers. Foundations of Physics **16**(6), 507–531 (Jun 1986). https://doi.org/10.1007/bf01886518
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional (1994)
10. Gemeinhardt, F., Garmendia, A., Wimmer, M.: Towards model-driven quantum software engineering. In: 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE). pp. 13–15. IEEE (2021). https://doi.org/10.1109/Q-SE52541.2021.00010
11. Grosan, C., Abraham, A.: Intelligent Systems: A Modern Approach. Springer Berlin Heidelberg (2011). https://doi.org/10.1007/978-3-642-21004-4
12. IBM: IBM Quantum. https://quantum-computing.ibm.com/, accessed: 26-10-2022
13. Magaz: samu-magaz/quantum-factory-method: Quantum Factory Method v1.0.0 (Jan 2023). https://doi.org/10.5281/zenodo.7544539
14. Moret-Bonillo, V., Magaz-Romero, S., Mosqueira-Rey, E.: Quantum computing for dealing with inaccurate knowledge related to the certainty factors model. Mathematics **10**(2) (2022). https://doi.org/10.3390/math10020189
15. Piattini, M., Serrano, M., Perez-Castillo, R., Petersen, G., Hevia, J.L.: Toward a quantum software engineering. IT Professional **23**(1), 62–66 (2021). https://doi.org/10.1109/MITP.2020.3019522
16. Shvets, A.: Dive into Design Patterns. Refactoring.Guru (2021)
17. Zhao, J.: Quantum software engineering: Landscapes and horizons. arXiv preprint arXiv:2007.07047 (2020). https://doi.org/10.48550/arxiv.2007.07047

ICCS Camera Ready Version 2023
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-36030-5_6