Advances in Engineering Software 173 (2022) 103216

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Engineering the development of quantum programs: Application to the Boolean satisfiability problem

Diego Alonso [*], Pedro Sánchez, Francisco Sánchez-Rubio

Division of Systems and Electrical Engineering (DSIE), Universidad Politécnica de Cartagena, Spain

ARTICLE INFO

Keywords:
Quantum computing
Model-driven engineering
Boolean satisfiability

ABSTRACT

The development of quantum programs is becoming a reality due to the rapid advancement of quantum computing. Over the past few years, a multitude of hardware platforms, algorithms, and programming languages have emerged to support this paradigm. By the very nature of Quantum Mechanics principles, there is an enormous change of philosophy when building quantum programs, which operate in a probabilistic space, unlike the deterministic behaviour shown by classical programming languages. These conceptual differences can be overcome by using techniques and tools of Software Engineering. In this paper, we apply Model-Driven Engineering techniques in a systematic way to ease the generation of quantum programs and we apply it to solve the satisfiability problem, very important in many engineering domains like verification of discrete systems and test of integrated circuits. To that aim, we contribute with a metamodel for representing quantum circuits and a model-to-text transformation to generate working IBM Qiskit code. This model-driven infrastructure is employed to automatically generate quantum programs from SAT equations through a model-to-model transformation that embeds Grover's algorithm. Besides, we provide formulas for calculating the number of required quantum elements from SAT equations, crucial in the current context of limited quantum resources. The interoperability with other tools and the extensibility to target additional quantum platforms is guaranteed thanks to the use of a model-based toolchain. We cover several usage scenarios to validate the approach, providing exemplary SAT equations, the generated Qiskit code and the results of executing this code in IBM Quantum infrastructure.

1. Introduction

Quantum Computing (QC) is a model of computation that employs Quantum Mechanics principles to perform a given computation. From an application point of view, the publication in 1994 of a paper where Peter Shor described a quantum algorithm that could break RSA encryption [1] in linear time, and the publication of Grover's algorithm for searching in an unordered dataset in constant time [2] radically opened the field of QC and attracted a lot of attention to it, by demonstrating that it could be applied to a wider range of computational problems than "just" the simulation of physical and chemical systems. Some of the application fields where QC can be applied include, but are not limited to, the following ones [3]: privacy and cryptography, supply chain and logistics, chemistry, economics and financial services, energy and agriculture, medicine and health, defence and national security programs, among others. Therefore, it is not surprising that many companies and governments are attracted by the business and strategic opportunities offered by quantum technologies [4].

QC can be expressed in several ways, but the circuit representation is the most usual one. Here, the programmer designs circuits that produce final states capable of revealing useful information about the problem at hand. QC relies on reversible operations, which transform the initial state of qubits into its final form by using only operations whose action can be inverted, and also on quantum superposition and quantum entanglement, phenomena that do not exist in classical computers. This way of working is radically different from what is done in classical computing. Quantum algorithms are then implemented as transformations acting on a complex vector space. The existing constraints when building quantum programs (unitarity of quantum operations, and the impossibility of non-intrusive measurement) make it quite complicated to conceive new quantum paradigms starting from existing classical ones. In the light of all this, it is of great importance to explore the possibility of bridging the gap between the way in which quantum algorithms are implemented and, at the same time, to take advantage of

Abbreviations: QC, Quantum Computing; MDE, Model-Driven Engineering; SAT, Boolean Satisfiability; CNF, Conjunctive Normal Form.

* Corresponding author.
E-mail address: diego.alonso@upct.es (D. Alonso).

https://doi.org/10.1016/j.advengsoft.2022.103216
Received 11 January 2022; Received in revised form 3 March 2022; Accepted 23 July 2022
Available online 11 August 2022

the good results obtained in Software Engineering (SE) over the last decades through the adoption of automatic software generation techniques.

All the advances in QC urge the SE community to contribute theories, methodologies and tools to support the proper development of quantum software, given that quantum computers are becoming more and more powerful and quantum supremacy is closer to be achieved every day. We are entering a new *Golden Age* [5] in Computer Science, where SE has to develop a specific body of knowledge for developing quantum software with all the quality attributes and best practices researched over the last forty years [3].

To this aim, in this paper we contribute a method to use the well-known Model-Driven Engineering (MDE) paradigm [6] that can greatly improve the development of quantum programs, and we apply it to solve the, also well-known, Boolean Satisfiability (SAT) problem [7]. On the one hand, MDE enables to raise the level of abstraction of the software that is being designed, putting the focus on the problem-domain concepts instead of on the solution-domain ones, which helps developers to better model the problem and to use generation techniques to automatically produce implementation code. This is especially needed for generating quantum software, given the large conceptual gap that exists between QC and classical computing. SAT, on the other hand, is a very important problem in many application domains, like the verification of discrete systems and the test of integrated circuits [8], and one of the first ones that has been proved to be NP-complete [9], meaning that its complexity grows exponentially with its size. This problem has been selected on purpose, since it makes no sense, in our opinion, to develop a quantum program for solving problems that a classical computer could easily solve. The intrinsic characteristics and potential applications of SAT makes it the ideal problem to show the benefits of applying MDE to develop quantum software.

To our knowledge, there are no research papers fully demonstrating the applicability of MDE to develop quantum programs, except some initial works outlined in Section 2.2, that propose metamodels for the last phases of quantum code generation, despite the increasing interest on adopting SE techniques for the development of quantum programs.

The rest of this paper is organized as follows. Section 2 describes the state of the art of the application of SE to QC and some known solutions for implementing quantum programs for solving SAT problems. Section 3 presents a metamodel for quantum circuits and a model transformation that generates Qiskit code, which can be run on IBM's Quantum infrastructure. Section 4 details the approach followed in this paper for automating the generation of quantum programs for solving SAT problems, which is based on a model transformation that generates a quantum circuit that embeds Grover's algorithm. Section 5 is devoted to validate the proposed approach and the tools developed. Section 6 includes a discussion on the motivational aspects and main concerns regarding the approach described in this paper. Lastly, Section 7 presents the conclusions and outlines some future research lines.

2. State of the art

This section includes an overview of the three main topics covered in this paper: software engineering for quantum computing, model driven engineering methods, and the main concepts behind the SAT problem definition and its implementations in quantum computers.

2.1. Towards quantum software engineering

Without any doubt, the quantum era has arrived. Quantum computing is more than a dream. But as with any new technology giving its first steps, quantum software developers will need standards, methods, and techniques to deal specifically with the singularity of the QC paradigm. One of the main challenges comes from the change from traditional bits to qubits, which can be physically created and managed by using different technologies.

A qubit can be in a superposition of the basic states $|0\rangle$ and $|1\rangle$, where the probabilities of a qubit being on a given state, once it is measured, are specified by complex numbers. By the very nature of Quantum Mechanics principles, the concrete values of these probabilities are, in general, unknown to the programmer, who will measure a classical '0' or '1' once the qubit is collapsed. This implies an enormous change of philosophy when building quantum programs because software development is oriented in QC towards the exploration of the problem space, searching for optimal solutions in a probabilistic space, unlike the deterministic behaviour shown by widespread classical programming languages. Thus, the state of a quantum system is determined by a vector in a complex vector space. Quantum programs are transformations acting on this vector space, following the axioms of Quantum Mechanics. One usual way to build up quantum programs is by means of circuits including quantum gates [10], which manipulate the qubits by changing their magnitude, phase, or both, in order to perform a given computation. A review of current quantum programming languages can be found in [11].

Just with a quick exploration of the scientific literature on Quantum Software Engineering, it is straightforward to realise that, nowadays, two different communities of computer scientists can be identified: on one hand, the community of quantum computer scientists with low or no background on SE, but highly skilled building quantum programs and thinking on a probabilistic way using algebraic notations; on the other hand, the community of SE researchers who are commonly too far from the quantum concepts, but with a strong background on tools and methods to support the classical software development process. Right in the middle, very recently, a few researchers, aware of the mistakes of the past for software development, are convinced that the adoption of a more agile approach for developing quantum programs and the importance of empirical validation are both key for a successful transition between both worlds [12].

Undoubtedly, the creation and boost of a new field, the recently named Quantum Software Engineering, must come from initiatives at different levels, among which training at university level is undoubtedly the most urgent one [3]. In this vein, the Talavera Manifesto for Quantum Software Engineering [12] is the result of recent discussions of academia and industry, where principles and commitments about how to adopt SE in quantum software development are identified. SE can contribute to quantum software development after agreeing on a set of principles and methodologies taken from experience. The Manifesto is a call to action to those stakeholders who should be involved in the process, mainly, software practitioners, researchers, educators, governments and funding bodies, quantum technology vendors, professional associations, among others.

One of the most complete surveys that tries to bring the classical concepts of SE to the quantum paradigm is [13] where, apart from some challenges and opportunities in the field, the quantum software life cycle is described, including the quantum software requirements analysis, design, implementation, test, and maintenance phases in quantum programs development. The paper also gives the first steps to close the gap between the classical and the new paradigm by proposing a set of extensions to UML for facilitating the modelling of quantum software. Through class and sequence UML diagrams, modellers can visually represent the different parts of a quantum program and the sequence of actions carried out on the working qubits.

In terms of the software process [14], provides a Quantum Development Life Cycle model to devise a set of systematic and cost-effective techniques to a successful quantum software development, detailing the different considerations to have in mind when defining each of the phases of quantum development process. Of paramount importance in the SE field is the definition of the basic unit of reuse. To this aim [15], discusses the concept of quantum module and establishes some rules for determining the cohesion and coupling levels of a quantum module. Related to this [16], introduces some ideas for facilitating the development of quantum software modelling languages by providing a conceptual model for quantum programs.

Considering the above, there is a high interest on giving the first steps on taking advantage of the previous experience on SE for classical computers and start building new solutions compatible with the quantum paradigm singularity. However, the results are still very incipient, with little or no concrete solutions beyond general intentions or approaches. That is why this paper represents a first step towards a concrete solution that is also applied to the resolution of a problem that is well-known in both classical and quantum computation.

2.2. Model-Driven engineering and its application to quantum computing

Over the last two decades, it has been demonstrated that the construction of software can greatly benefit from the adoption of the well-known MDE approach [17]. The most important motivation to adopt MDE is improving productivity both in terms of the increase in the software artefact's value and the longer expectation of use [18]. The more functionality you can derive from the models, the higher the productivity you will obtain. In MDE, the primary focus of software development are models and the transformations between them. A model is a reduced version of the represented system where some details are hidden or removed, given that they are irrelevant from a given point of view. At the same time, models are a way to share knowledge among technical and non-technical stakeholders. The direct advantage of MDE is that software engineers can express models using concepts closer to the problem domain. In this way, models can be specified and manipulated easier, and they are less dependent on computing technology and underlying execution platforms. Of course, in the end, computer programs should be automatically obtained from their corresponding input models. Characteristics such as understandability, accuracy, predictiveness and inexpensiveness should be taken into account when designing models [19]. Models conform to their metamodel, meaning that a model must satisfy the concepts, relationships and constraints established at the metamodel level. At the same time, a metamodel is "a model that defines the structure of a modelling language" [20].

A model transformation is a set of rules to transform an input model (conforming to a source metamodel) to an output model (conforming to a target metamodel). A transformation engine is then a tool that uses model transformation rules to produce output model(s) from input model(s). Model transformations include both model-to-model transformations, with which input models are used to obtain different kind of models, probably in different languages and abstraction levels; and model-to-text transformations, which automate the transformation of models to textual representations such as code or documentation. A comprehensive survey addressing of the classification of the known model transformation approaches is given in [21]. The automated tasks can also include the verification of models to analyse them for some desirable properties and the absence of those to be avoided [22].

Regarding the application of modelling techniques to develop quantum programs [23], provides a solution to model quantum circuits by means of a UML profile applied to the UML activity diagram. This greatly facilitates the integration of quantum models with classical ones, favouring the development of hybrid applications. In [24], the authors argue for researching on how MDE may be applied for quantum technologies. Particularly, they advocate for an approach for the development of hybrid applications and point out to the most recent proposals in this field. Of particular interest is the framework proposed in [16], where some ideas for developing quantum software modelling languages are presented. More specifically, they consider an approach where there are metamodels for domain specific languages, as well as for modelling quantum programs as extensions of UML. With these metamodels, developers may model their applications including both classical and quantum programs. Related to these works, the authors of [25] propose MDE4QAI, a framework with a MOF-based metamodel for the integration of QC and Artificial Intelligence. They advocate enhancing the use of domain-specific languages to facilitate the development of hybrid programs.

These are, to our knowledge, the existing papers that apply modelling techniques to QC. It is evident that we are just at the beginnings of contributing to QC using SE techniques and methods.

2.3. Quantum approaches to solving the Boolean satisfiability problem

The Boolean satisfiability problem can be expressed as follows: given a logical equation, determine if there exists a Boolean assignment to its variables which makes the equation true. In general, the equation can make use of any logical operator and SAT can be extended to include quantifiers (like '*for all*') and even predicates over variables and functions, which define the Satisfiability Modulo Theories (SMT) problem [26]. This paper focuses on the Boolean SAT that employs only the *AND, OR, NOT* operators in conjunctive normal form (CNF), since this is the form usually used to solve SAT problems [9].

A logical equation in CNF is expressed as a conjunction (logical *AND*) of a set of clauses, where each clause is defined as a disjunction (logical OR) of literals, where each literal has the form x or $\neg x$. For example, the formula $(\neg a \vee b) \wedge (\neg a \vee b \vee c)$ is expressed in CNF. In the 3-SAT problem, each clause has at most three literals. It has been demonstrated that any k-SAT problem can be transformed to a 3-SAT one [9], and therefore there is no loss of generality in focusing on the Boolean 3-SAT problem. The SAT problem has been extensively studied since the early seventies. Many algorithms have been developed in order to solve it and it is widely used in many application domains, as described in [7].

There are several strategies to apply QC to solve the SAT problem. The most well-known ones are those that rely on the application of Grover's search algorithm [2] to find solutions, if they exist. The basic description of the application of Grover's algorithm to the SAT problem can be consulted in [27]. The application of this algorithm requires (i) an *oracle*, which can distinguish between correct and incorrect solutions (assignments to the logical variables in this case), and (ii) the inclusion of the *Grover* diffuser, a quantum circuit that amplifies the probabilities of correct solutions. Both the oracle and the diffuser circuits constitute the *Grover iteration*.

If the oracle is constructed naïvely, that is, by directly translating the AND, OR, NOT logical operators into their equivalent quantum gates, the resulting quantum circuit has, however, an important drawback: the number of required qubits increases linearly with the number of disjunctions that appear on the logical equation, since one additional qubit is needed to store the superposition result of each of them. For instance, the logical equation shown before, $(\neg a \vee b) \wedge (\neg a \vee b \vee c)$, requires 5 qubits to be solved: 3 for the variables (a,b,c) and 1 for each disjunction. Thus, we have almost doubled, in this particular example, the number of input qubits required by the equation. Qubits are not only scarce resources, but they are also difficult to manage and control with currently available technologies. Moreover, the greater the number of qubits, the greater the probability of suffering de-coherence effects while executing the computation, which can ruin the result [28].

Some research works focus on modifying the application of Grover's algorithm in order to improve some properties of the resulting quantum program, such as the number of required qubits, the number and types of the employed quantum gates, the number of steps required to solve the problem, etc. In [29], an overview of searching algorithms that are commonly applied in QC, describing an application to the 3-SAT, is provided. Cheng [30] presents a proposal where they solve a smaller version of the SAT problem by employing well-known classical algorithms, and then apply Grover's algorithm to solve the remaining variables. More recently, Wang [31] describes a solution to the 3-SAT that combines the non-quantum algorithm developed by Schoning [32] with a step of amplitude amplification to reduce computation time and the number of qubits and quantum gates needed to solve it.

Furthermore, there are some research works that focus on other properties or other ways of exploiting Quantum Mechanics principles to perform computation. The first work we highlight is [33], which employs the adiabatic model of QC. This very mathematical model uses Hamiltonian matrices to model the quantum evolution of a system. The authors of the paper describe how to modify the computation to be able to start it from an initial guess of the solution to the SAT problem, so that the evolution of the adiabatic model itself "guides" the computation to find a possible solution to it. The more recent work [34] employs continuous-time quantum walks to find a solution to K-SAT problems. Such algorithm is also based on the use of Hamiltonians.

As can be seen, quantum solutions for SAT problems have been properly analyzed, enough to consider them as an appropriate case study for testing an MDE solution to automate the generation of quantum programs. The following sections describe the approach proposed in this paper.

3. A metamodel for quantum circuits and quantum code generation

We need, as a first asset that enables supporting the approach, a metamodel to model quantum circuits, as it is the most usual representation of quantum computations. After that, a model-to-text transformation is required to generate working code for a quantum computer or simulator.

To our understanding, when representing quantum circuits using metamodels, two main options can be identified. The first one is the approach oriented to represent the flow of control given as a sequence of connected actions (gates), obtaining as a result a fully connected acyclic graph. This is the solution adopted by [23] and may have some benefits in terms of optimization of quantum algorithms. The second approach, which is the one chosen in this work, sees the circuit as a sequence of ordered vertical slices in which single or complex gate operations can be included. This way of modelling has the advantage of identifying very directly the set of gates operating on the qubits without the need of traversing a graph. Furthermore, additions or removal of gates when editing the circuit are more straightforward.

According to this, the intuitive idea behind the metamodel proposed in this paper, entitled Qcore and shown in Fig. 1, is dividing the circuit in vertical slices. Each `circuit` has associated a set of input and `ancilla qubits`, and a set of classical (output) `bits`. Each `Slice` can be either a `Barrier` (used only for printing the circuit) or a `QSlice` containing operations. `Operations` are sub-classified into `Measure`, basic `Gate`, `Controlled operation`, and `Reset` operations. As subclasses of Gate, all the different quantum gates of a quantum computer can be identified. Each of these gates has a link to the qubit in which the operation is performed.

The `SWP` (swap) gate is a special one as it also stores the second qubit with which the swapping operation is performed. Measure and reset operations reference the qubit in which the operation is performed. The `Controlled operation` is the most complex one. It allows to model zero or many control-qubits acting on one or more basic gates. This is the way to model, for instance, the CNOT gate (one qubit as control and one qubit targeted with a single gate), the CCNOT gate (two qubits as control and one as target), as well as other combinations such as CCNOTNOT

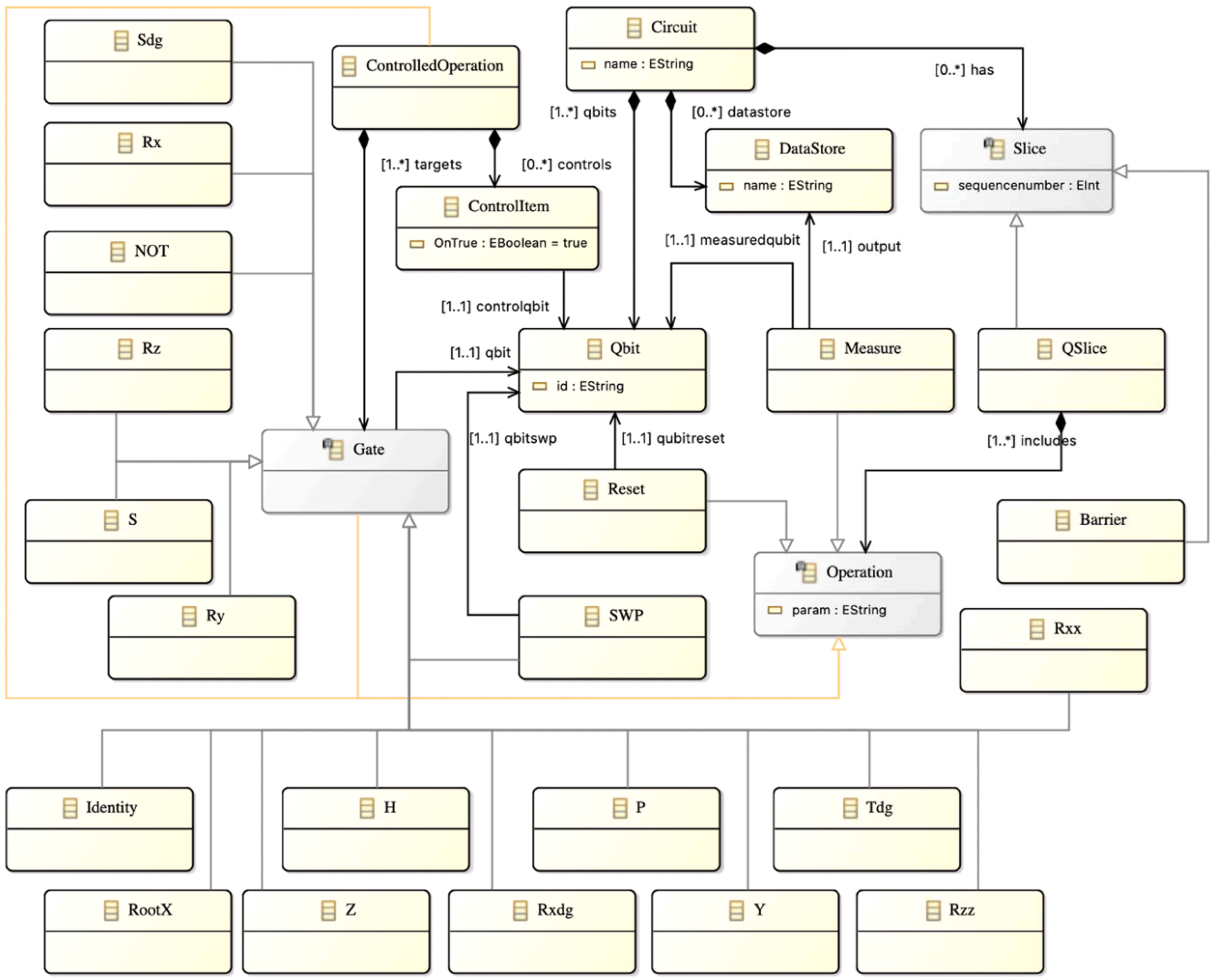


Fig. 1. Qcore metamodel for modelling quantum circuits.

(that is, two control qubits acting on two NOT gates). The specific number of basic gates can be extended as needed depending on the specific platform.

From this general-purpose metamodel, which allows us to model quantum circuits, it is possible to generate code for almost any quantum programming language. In our case, we have decided to target the Qiskit SDK [35] language by IBM, since it is one of the most well-known. Given the structure of the Qcore metamodel, the Qcore-to-Qiskit model-to-text transformation is quite straightforward. Fig. 2 shows an excerpt of the main part of this transformation, which has been programmed in the Epsilon language [36]. As can be seen on the transformation source code, registers of qubits and classical bits are initialized according to the input model and a quantum circuit is then constructed (lines 1 to 3). After that, the slices conforming the circuit are processed, one by one. In case it is a QSlice, all the gates inside it are again processed (lines 4 to 13) by invoking a `to_qiskit()` operation that has been defined for each Operation. At the end of Fig. 2 we show the operation `to_qiskit()` created for the Hadamard gate (lines 15 to 18). Some more code is added in order to generate a working Qiskit source file, like `import` statements, printing the circuit and configuring and running the simulator, for instance.

4. Generation of quantum programs for solving SAT problems

Once the infrastructure for modelling quantum circuits has been created, it is possible to create a new metamodel, conceptually above the Qcore one, that enables us to model problems at a higher level of abstraction, in our case, the SAT problem. Thus, the application of MDE principles to generate quantum programs for solving the SAT problem comprises two metamodels and two model transformations, as shown in Fig. 3. This follows a classical MDE scheme, where the final result of the transformation chain is source code for IBM Qiskit SDK, which can be directly run on IBM's Quantum infrastructure.

The metamodel for modelling Boolean equations in CNF is shown in Fig. 4. As can be seen, it comprises three meta-classes: representation of the whole `Logical equation`, which is an AND operation of `Clauses`, which in turn are an OR operation of `Atoms`, which can be negated.

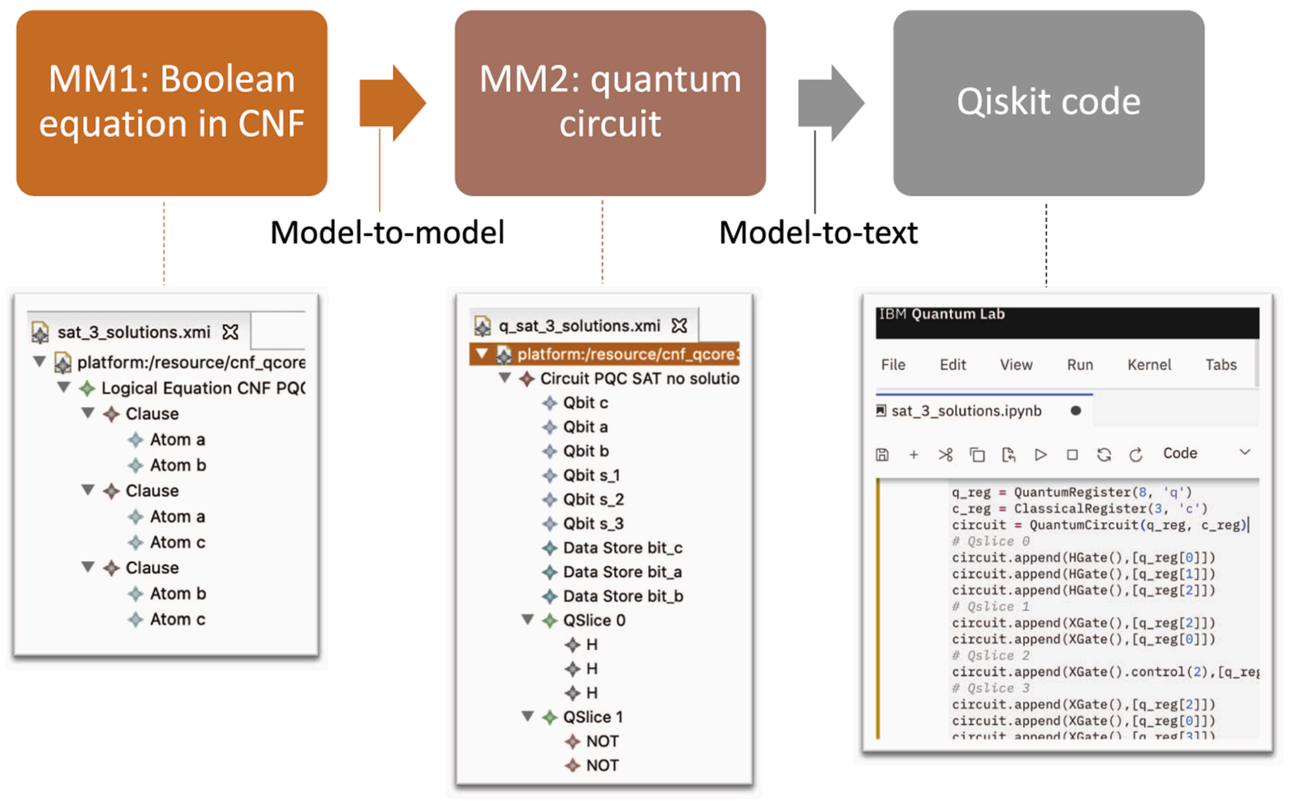
The model transformation that generates the quantum circuit for solving a given SAT problem follows the scheme described in [27]. Being SAT a search problem, it is natural to apply Grover's algorithm [2] to solve it. This algorithm includes the generation of an oracle to mark correct solutions and a Grover diffuser to amplify their probabilities. If the search space has size N and there are M solutions to the SAT equation, both the oracle and the Grover diffuser have to be iterated $O(\sqrt{N/M})$ times in order to obtain a solution to the search problem with high probability, as described on [37]. In the case where M is not known, some approaches can be adopted to make an estimation of its value, such as quantum counting [38]. An excerpt of the model-to-model transformation, where it is possible to identify the creation of the main parts of the quantum circuit as described in the following paragraphs, is shown in Fig. 5.

The CNF-to-Qcore model-to-model transformation starts by creating a qubit and a measure for each atom, and adding a Hadamard gate to each input qubit to put it in superposition state (line 1 in Fig. 5), which is a basic quantum state. Putting qubits in a superposition state is one of the first operations performed on every quantum program. Almost all quantum manipulation operations are performed on qubits in this state, in which all possible combinations of $0'$s and$1'$s are explored at the same time thanks to quantum principles. The oracle is then constructed as follows: for each clause, we create the quantum version of the OR operation over the corresponding qubits (by adding a controlled-NOT gate), and one ancilla qubit to store its result. Ancilla qubits are commonly used on QC for storing temporal values generated as part of the computation. In case any of the input atoms is negated, we need to also negate the qubit by adding a NOT gate before and after performing the OR operation. The second NOT gate is needed because we need to restore the qubit to its original state after manipulating it. All the `Qslices` created to store the quantum gates generated by the aforementioned transformation constitute the oracle, and are added to the output quantum circuit (line 2 in Fig. 5). Lastly, we finish constructing the oracle by adding a controlled-Z gate that acts on all ancilla qubits, flipping the phase of all combinations that are true, and then add it to the output circuit (line 3 in Fig. 5).

The circuit is not finished yet since we first need to undo all the modifications applied to the input qubits, and lastly apply the Grover diffuser. The undo operation is needed in order to preserve the reversibility properties of QC and requires us to reverse the order of execution of the oracle circuit. Fortunately, this is easily achieved by using the

```
1  q_reg = QuantumRegister([%=circuit.qbits.size()%], 'q')
2  c_reg = ClassicalRegister([%=circuit.datastore.size()%], 'c')
3  circuit = QuantumCircuit(q_reg, c_reg)
4  [%for (s in circuit.has) { %]
5    [%if (s.isTypeOf(Barrier)) { %]
6      circuit.barrier(q_reg)
7    [%} else { %]
8      # Qslice [%=s.sequencenumber%]
9      [%for (g in s.includes) { %]
10       [%=g.to_qiskit()%]
11     [%}%]
12   [%}%]
13 [%}%]
14 [% @template
15 operation H to_qiskit()  {
16   var data = self.data();%]
17   circuit.append([%=data.first()%], [q_reg[[%=data.second()%]]])
18 [%}
```

Fig. 2. Excerpt of the Qcore-to-Qiskit model-to-text transformation, written in Epsilon Generation Language (EGL), focusing on the overall generation of Qiskit code.



```
q_reg = QuantumRegister(8, 'q')
c_reg = ClassicalRegister(3, 'c')
circuit = QuantumCircuit(q_reg, c_reg)
# Qslice 0
circuit.append(HGate(),[q_reg[0]])
circuit.append(HGate(),[q_reg[1]])
circuit.append(HGate(),[q_reg[2]])
# Qslice 1
circuit.append(XGate(),[q_reg[2]])
circuit.append(XGate(),[q_reg[0]])
# Qslice 2
circuit.append(XGate().control(2),[q_reg
# Qslice 3
circuit.append(XGate(),[q_reg[2]])
circuit.append(XGate(),[q_reg[0]])
circuit.append(XGate(),[q_reg[3]])
```

Fig. 3. Scheme of the MDE process: from CNF logical equations to Qiskit source code.

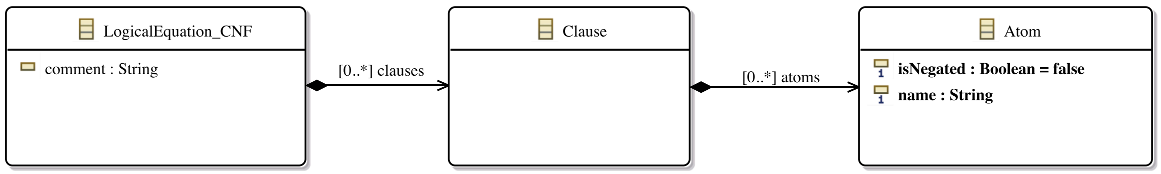


Fig. 4. Metamodel for modelling SAT problems in CNF.

```
1  the_circuit().has.add(hadamard_input_qbits());
2  oracle.forEach(i | the_circuit().has.add(i));
3  the_circuit().has.add (phase_flip());
4  var oracle_copy = emfTools.ecoreutil.copyAll(oracle);
5  oracle_copy.invert().forEach(i | the_circuit().has.add(i));
6  var grover = grover_mirror();
7  grover.forEach(i | the_circuit().has.add(i));
8  the_circuit().has.add(measure_qslice);
```

Fig. 5. Excerpt of the CNF-to-Qcore model-to-model transformation, written in Epsilon Transformation Language (ETL), focusing on the overall generation of the quantum circuit.

`invert()` operation available in Epsilon (lines 4 and 5 in Fig. 5). After that, we only need to add the Grover diffuser (lines 6 and 7 in Fig. 5), which is a set of gates that operate over the input qubits, and the measure operations (line 8 in Fig. 5). It is possible now to provide a non-quantum explanation on how Grover's algorithm works:

1. By putting all input qubits into superposition state, we prepare them to take all possible combinations of 0′s and1′s at the same time. Thus, we are capable of exploring 2^n combinations running the circuit once.
2. The oracle then flips the phase (a qubit state is specified by complex numbers and thus a qubit state is characterized by magnitude and phase) by 180° of all the combinations of 0′s and1′s that make the oracle answer *true*. This is why the oracle implements the logical SAT equation.
3. The Grover diffuser converts phase differences into magnitudes (the only value that affects the probability of reading a 0 or 1 when

measuring a qubit), amplifying the magnitudes of the combinations that have been flipped, therefore making the probabilities of the qubits collapsing to those combinations, which are answers to the problem, higher.

4. The Grover iteration has to be repeated $O(\sqrt{N/M})$ times in order to increase the probability of obtaining a valid solution.

The transformation scheme followed for implementing Grover's algorithm is the general one, as described in the state of the art. Thus, the transformation does not consider potential optimizations that could be performed in order to (i) reduce the number of gates of the generated quantum circuit (for instance, in the concrete quantum circuit shown in Fig. 6, it is possible to cancel out two consecutive NOT gates on qubit q_0), (ii) reduce the number of required qubits, for instance, by applying Grover's algorithm to a subset of the variables involved in the SAT equation, as described in [30].

As shown in this section, the use of well-known best practices, methods and tools in SE can definitely ease the implementation of quantum programs. Given the facilities provided by the MDE approach, it is possible to develop and include new tools that optimize the generated QCore model before generating the Qiskit code. In this way, the contribution of MDE to ease the development of quantum programs is twofold: on the one hand, it provides a higher abstraction level where the developer can use the concepts of the problem domain (Boolean formulas in this case) instead of the concepts of the solution domain (quantum gates) and on the other hand, it minimizes potential errors introduced by programmers when they directly encode the solution in the quantum programming language. For instance, as described in the following section, the quantum circuit generated by a Boolean formula that includes five clauses with nine atoms comprises sixty-nine quantum elements (both qubits and gates). Therefore, MDE can make the design and development process significantly more efficient and cost-effective through automation.

Next section is devoted to validating the transformations through some representative examples covering distinct situations. All the metamodels, example models and tools described in the paper are available in the GitHub repository [39].

5. Validation of the approach

Testing model-to-model transformations has been broadly researched in the literature [40], where three main challenges are commonly identified [41]: the creation of a set of input test models, the definition of adequacy criteria for checking whether the input models are sufficient for the testing task, and lastly, the verification that the generated models are actually the expected ones. In this regard, there have been many approaches providing solutions for these challenges, like unit testing [42], mutation analysis techniques [43], or static analyses [44], to mention a few. Despite these contributions, there is little consensus in how to adequately validate model transformations given the broad spectrum of approaches to model testing as well as the intrinsic difficulty of this task. Because of the difficulty to assure that a set of input test models satisfies all the constraints required for a concrete test, automatic test model generation is not usually performed, but rather test models are manually created. Since this task is also difficult, error-prone and tedious, model transformations are commonly best validated by checking a set of properties on the output model in order to determine if the transformation has been correctly performed.

In face of it, we have implemented a validation approach that revolves around two types of tests: (i) check the correctness of the generated quantum circuits by running them using the facilities provided by IBM and the Qiskit SDK, and (ii) verify the output model in terms of the expected number of quantum elements to be generated. The description of the validation tests follows.

5.1. First test

The first test would involve checking that all possible types of input models, SAT equations in CNF in our case, generate valid quantum circuits in the Qiskit language, and that these circuits solve the input SAT

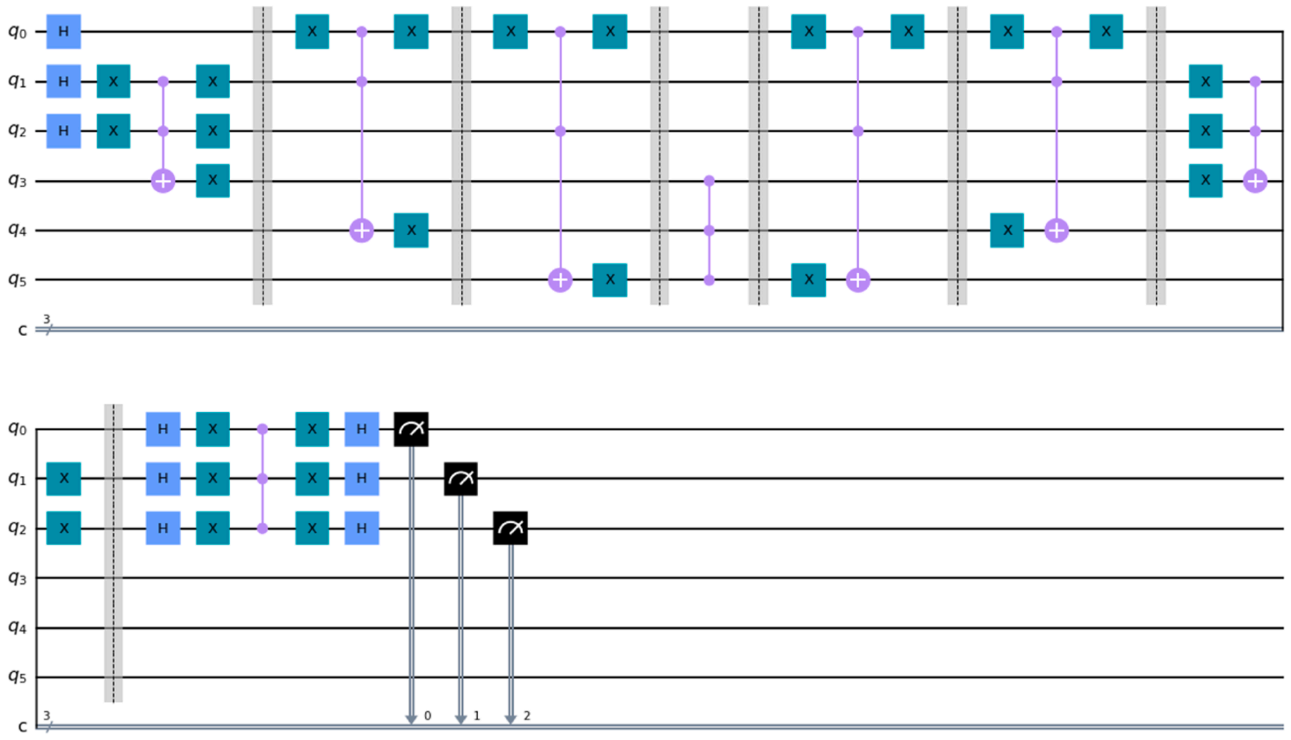


Fig. 6. Quantum circuit for solving the SAT logical equation with three solutions, as generated by IBM Quantum Computing Lab online tool. Each vertical set of gates or barrier is modelled as a slice on the Qcore metamodel.

problem. Since this is impossible for obvious reasons, we have manually generated and tested a variety of input CNF models in order to achieve a complete coverage of the most representative scenarios, i.e. equations with multiple clauses and atoms appearing in all of them and in a subset of the clauses. Given that the number of qubits needed to implement the CNF equations grows proportionally with the number of clauses and atoms (more details about this on Section 5.2), there is a real limitation in the size of the formulae that can be tested (at the time of writing, measured in tens of qubits). Considering this, and in order to ease the presentation of the results of the approach, we focus on three concrete equations, taken from [27], that have a different number of solutions, ranging from multiple solutions to none. After this, we present the results of solving three larger equations with varying number of atoms and clauses, up to 20 qubits, which is the current limitation imposed by the simulator available on IBM's Quantum infrastructure.

All the generated circuits have been run on IBM's Quantum infrastructure, running for each of them 1024 simulations (see Section 5.3 for more details about the number of simulations). This is another characteristic that radically differentiates QC from classical computing: given the probabilistic nature of the manipulation of the circuit qubits, it is necessary to run the quantum circuit hundreds or thousands of times to obtain a probability distribution of the possible solutions to the problem. The combinations that have higher probabilities (see Fig. 9) are the candidates to be real solutions of the problem, but you cannot be completely sure since QC operates in a probabilistic space. The equations are the following ones:

- Equation with three solutions (SAT #1): $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_3)$. Solutions are $x_1 = F, x_2 = T, x_3 = T; x_1 = T, x_2 = F, x_3 = T; x_1 = T, x_2 = T, x_3 = T$. The quantum circuit generated for this equation is shown in Fig. 6, while the results of the simulation are shown in Fig. 9a. As can be seen on the last figure, the three combinations with the highest frequencies in the histogram are the solutions to the equation.
- Equation with one solution (SAT #2): $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_3)$. The only solution is $x_1 = T, x_2 = F, x_3 = T$. The quantum circuit generated for this equation is shown in Fig. 7, while the results of the simulation are shown in Fig. 9-b. As can be seen on the last figure, the combination with the highest frequencies in the histogram is the solution to the equation.
- Equation with no solution (SAT #3): $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_3) \wedge (x_2)$. The quantum circuit generated is shown in Fig. 8, while the results of the simulation are shown in Fig. 9c. In the last figure, it is possible to see that the frequencies of the combinations shown in the histogram are roughly the same, meaning than either all combinations are correct, or all are incorrect. After checking one of them, it is possible to conclude that are all incorrect. This behaviour is expected given the probabilistic nature of QC.

The more complex equations simulated comprise 15 atoms and 5 clauses, 10 atoms and 10 clauses, and 5 atoms and 15 clauses, respectively (considering the aforementioned limitation in the number of qubits available). These CNF equations are the following ones:

- 15 clauses and 5 atoms: $(x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_3 \vee x_4 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x_3 \vee x_4 \vee x_5) \wedge (x_2 \vee \neg x_4 \vee x_5) \wedge (\neg x_1 \vee \neg x_2 \vee x_3 \vee x_5) \wedge (x_2 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_3 \vee x_5) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5) \wedge (\neg x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5)$. The quantum circuit comprises 20 qbits and the following number of gates: 172 NOT, 30 CNOT, 2 CZ, and 15 Hadamard. 375 lines of Qiskit code have been generated.
- 10 clauses and 10 atoms: $(x_1 \vee x_2 \vee \neg x_3 \vee x_5) \wedge (x_6 \vee \neg x_7) \wedge (x_7 \vee x_8 \vee x_9 \vee \neg x_{10}) \wedge (x_1 \vee x_3 \vee x_5 \vee x_7 \vee x_9) \wedge (x_2 \vee x_4 \vee x_6 \vee \neg x_8 \vee x_{10})$

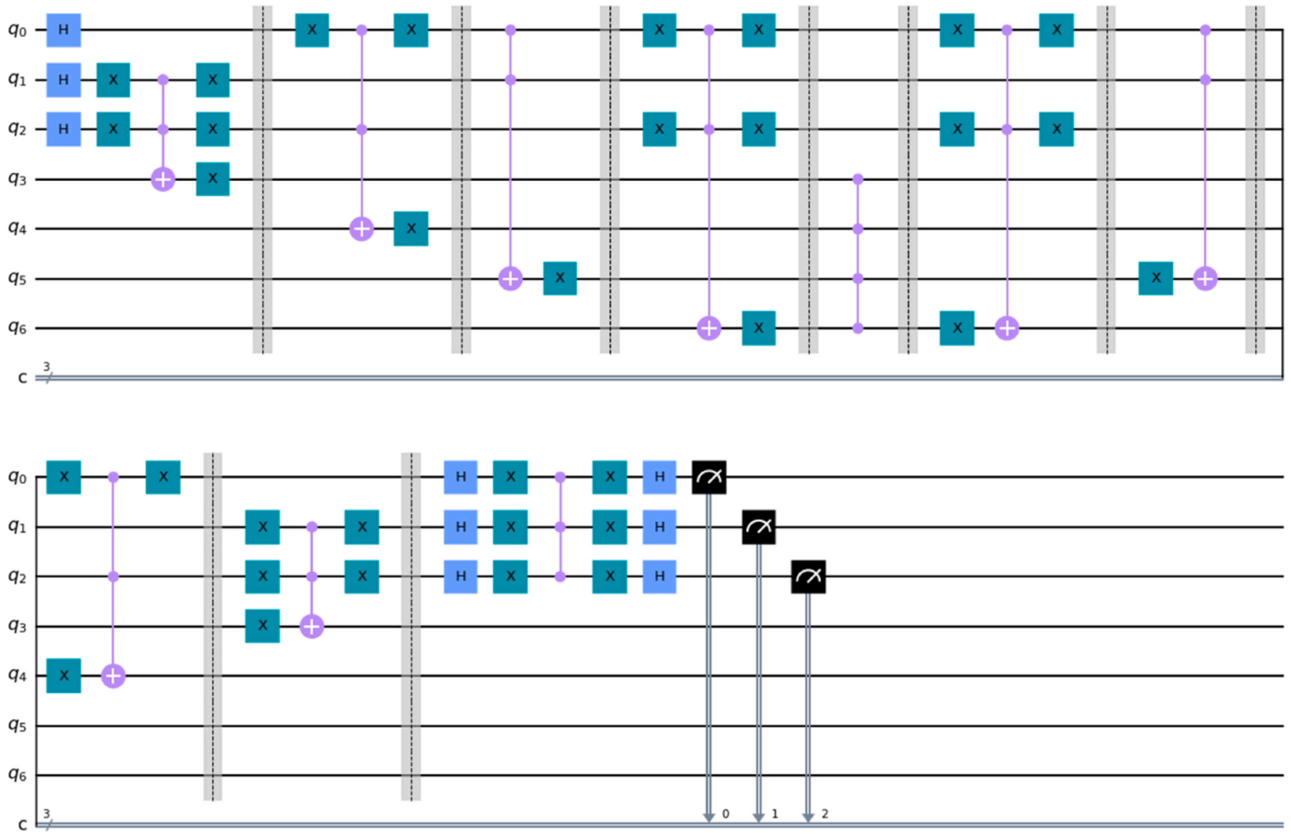


Fig. 7. Quantum circuit for solving the SAT logical equation with one solution, as generated by IBM Quantum Computing Lab online tool.

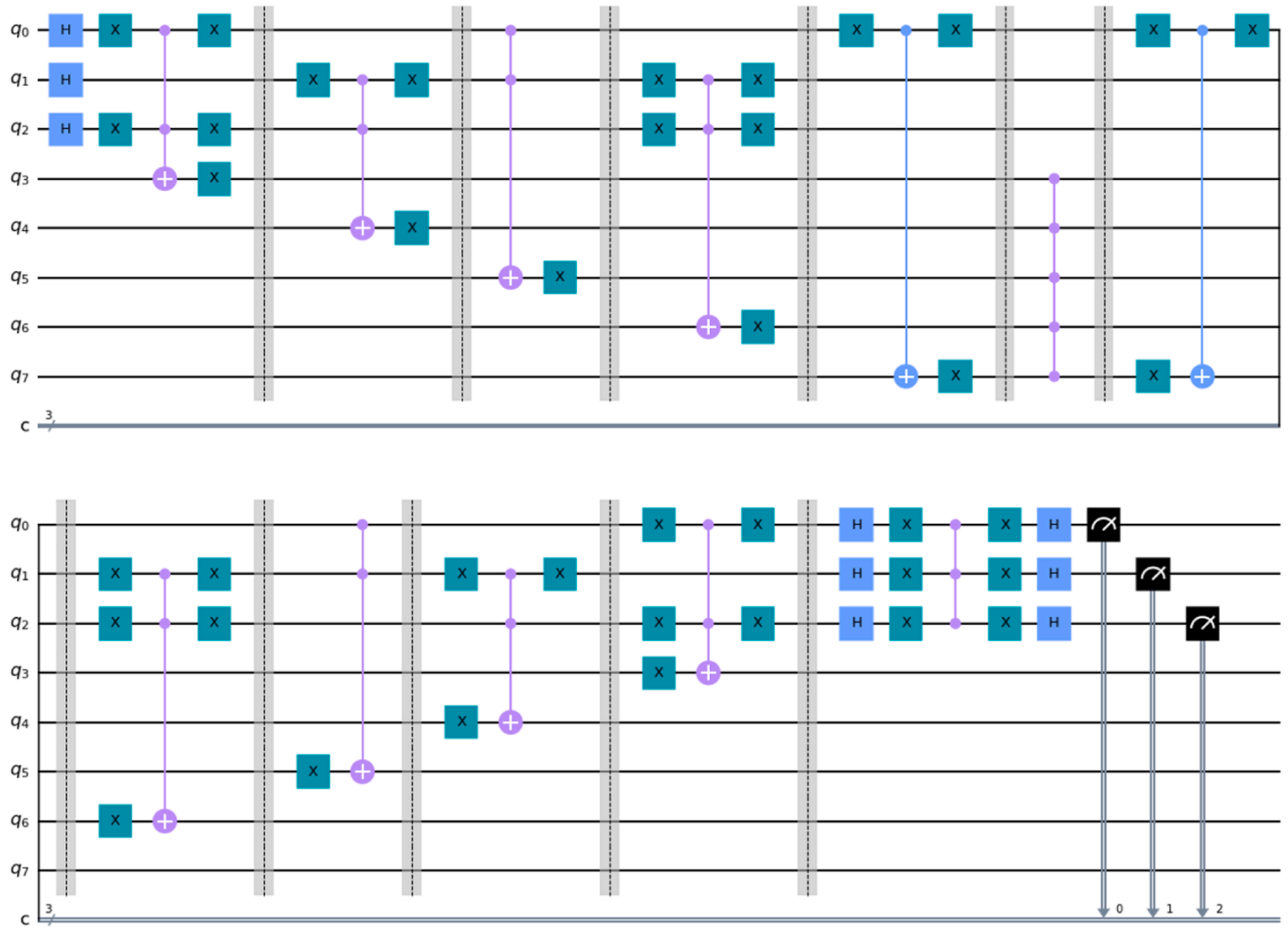


Fig. 8. Quantum circuit for solving the SAT logical equation with no solutions, as generated by IBM Quantum Computing Lab online tool.

$\wedge(\neg x_1 \vee x_2 \vee x_3 \vee x_4 \vee \neg x_5 \vee x_8 \vee \neg x_9 \vee \neg x_{10}) \wedge (x_2 \vee \neg x_3 \vee x_4 \vee \neg x_6 \vee \neg x_8) \wedge(x_3 \vee x_4 \vee \neg x_5 \vee x_7 \vee \neg x_8 \vee x_9 \vee \neg x_{10}) \wedge (\neg x_1 \vee x_3 \vee \neg x_8 \vee x_9 \vee x_{10}) \wedge(x_2 \vee \neg x_4 \vee x_6 \vee \neg x_7 \vee x_8 \vee \neg x_{10})$. The quantum circuit comprises 20 qbits and the following number of gates: 172 NOT, 20 CNOT, 2 CZ, and 30 Hadamard. 350 lines of Qiskit code have been generated.

- 5 clauses and 15 atoms: $(x_2 \vee x_4 \vee x_6 \vee \neg x_8 \vee x_{10} \vee x_{12} \vee \neg x_{13} \vee \neg x_{14}) \wedge(\neg x_1 \vee x_2 \vee x_3 \vee x_4 \vee \neg x_5 \vee x_8 \vee \neg x_9 \vee \neg x_{10} \vee x_{14} \vee \neg x_{15}) \wedge (x_2 \vee \neg x_3 \vee \neg x_6 \vee \neg x_8 \vee x_{11} \vee x_{13}) \wedge (x_3 \vee x_4 \vee \neg x_5 \vee x_7 \vee \neg x_8 \vee x_9 \vee \neg x_{10} \vee x_{12} \vee \neg x_{14}) \wedge(x_2 \vee \neg x_4 \vee x_6 \vee \neg x_7 \vee x_8 \vee \neg x_{10} \vee x_{13} \vee \neg x_{15})$. The quantum circuit comprises 20 qbits and the following number of gates: 128 NOT, 10 CNOT, 2 CZ, and 45 Hadamard. 270 lines of Qiskit code have been generated.

5.2. *Second test*

The aim of the second test is to check the number of quantum elements generated by the CNF-to-Qcore model transformations, according to the equations shown in Table 1. In the table, ν represents the number of variables (the set of variables is the set of distinct values of the property 'name' of the atoms), κ is the number of clauses, α is the number of non-negated atoms of the equation, and ρ is the number of repetitions of the Grover iteration. These equations have been integrated into the source code of the CNF-to-Qcore model transformation, so that we can verify that it generates the expected number of quantum elements. The table also shows the number of quantum elements generated for each of the three CNF equations. Regarding the scalability of the proposed approach, as can be seen in the table, the number of generated qubits and gates is linear with respect to the size of the input SAT equation, which will allow for solving large SAT problems, provided the available quantum infrastructure can run the generated quantum programs.

These equations allow us to further test the approach by using larger CNF equations as input models and checking that the QCore model contains the number of expected quantum elements. For this, we have developed a random model generator using the Epsilon Model Generator language. With this tool, we have tested several input models ranging from tens of clauses and atoms per clause, to almost a thousand clauses and a hundred atoms per clause. As an example, from a CNF equation comprising 951 clauses with between 30 and 99 atoms per clause, the model transformation generates a QCore model with 1051 qubits, 125,006 NOT gates, 1902 CNOT gates, 2 CZ gates and 300 Hadamard gates, as predicted by the equations shown in Table 1 with one Grover iteration. This QCore model is then transformed into a Qiskit file with around 135,000 lines of code, which cannot be simulated on IBM's Quantum infrastructure right now for the reasons already mentioned. The size of the generated Qiskit file shows another advantage of the approach, since it would be very easy for a human programmer to make a mistake implementing such a large quantum circuit.

Further tests would involve having a test suite to perform unit testing on the generated Qiskit code, similarly to what is done with Java code by using JUnit. However, there is currently no such unit testing framework for checking quantum code. This is another of the deficiencies of the QC field when compared to classical software development, as identified by [3], and one of the many areas waiting for contributions from the SE community.

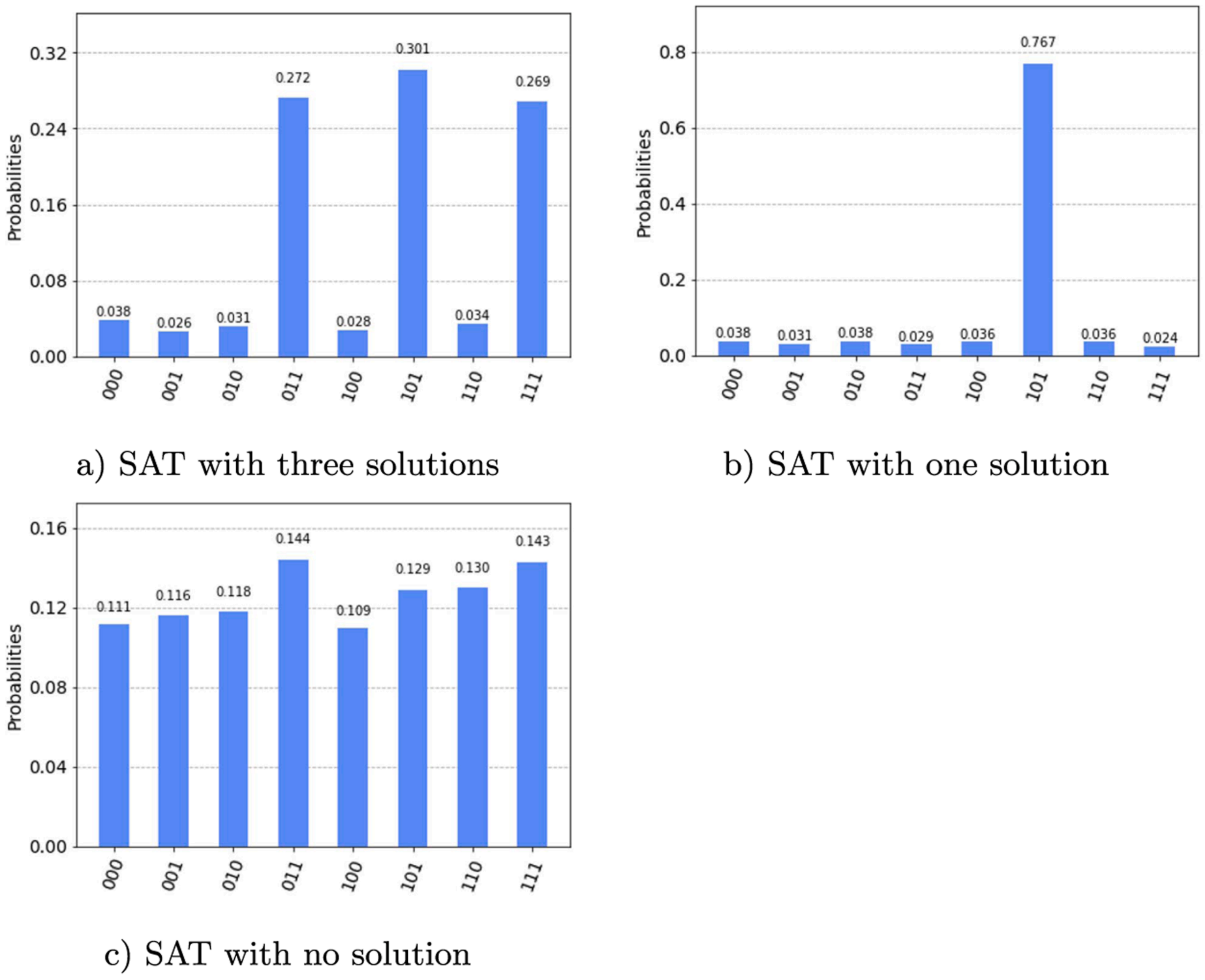


Fig. 9. Probabilities histogram of the solutions to each SAT equation.

Table 1
Number of quantum elements generated by the transformations for a SAT equation. ν is the number of variables (atoms with distinct names), κ is the number of clauses, α is the number of non-negated atoms and ρ is the number of repetitions of the Grover iteration.

Quantum meta-classes	# elements generated	SAT #1	SAT #2	SAT #3
qubit	ν	3	3	3
ancilla	κ	3	4	5
CX gate	$2{\cdot}\kappa{\cdot}\rho$	6	8	10
H gate	$(1+2{\cdot}\rho){\cdot}\nu$	9	9	9
CZ gate	$2{\cdot}\rho$	2	2	2
X gate	$2{\cdot}(\nu+\kappa+2{\cdot}\alpha){\cdot}\rho$	28	34	40

5.3. *On the number of simulations of the quantum program*

Given the probabilistic nature of QC, it is common to run a quantum program several times so that the probability distribution of the solutions obtained by running the program corresponds to the actual solutions to the problem. In this sense, the IBM platform allows the parameterization of the number of simulations run, which by default is set to 1024, and this is the number we have used. Several authors have approached the analysis of the probability of success of Grover's algorithm [38,45], which depend on the number of solutions (M), the size of the problem (N) and the number of Grover iterations. For the case of a single iteration, Eq. (1) determines the probability of not finding a valid solution with a single run of the circuit [45].

$$P_{failure} = 1 - 9{\cdot}\frac{M}{N} + 24{\cdot}\left(\frac{M}{N}\right)^2 - 16{\cdot}\left(\frac{M}{N}\right)^3 \tag{1}$$

A key question that arises is how many simulations (κ, an integer value) of the circuit should be performed to obtain at least a valid solution with probability greater or equal than a certain value ρ. Considering the above equation and performing basic statistical manipulation, κ is determined with the expression shown in Eq. (2).

$$\left(P_{failure}\right)^{\kappa} \leq (1-\rho) \rightarrow \kappa \geq \left\lceil \frac{\log(1-\rho)}{\log\left(P_{failure}\right)} \right\rceil \tag{2}$$

According to Eqs. (1) and (2), the probabilities of failure for equations SAT #1 and SAT #2, shown in Section 5.1, are 15.63% and 21.87%, respectively. Setting the probability of success to 95% for obtaining at least one valid solution, just two runs will suffice in both cases. Rising the probability of success to 99%, three and four runs are at least needed, respectively.

6. Discussion

The proposed application of the MDE approach to generate a quantum program to solve the SAT problem is very flexible and it is possible to extend it in order to make it more powerful or to adapt it to any particular need. Without aiming to provide a complete set of improvements, some further extensions that could be included on the basis of the infrastructure that has been developed are the following ones:

- Regarding the targeted quantum computer, the most obvious extension could be to develop a new model-to-text transformation to generate source code for other quantum computers, provided that the target computer supports the circuit representation of a quantum computation. The Qcore metamodel could be also extended by adding subclasses of the Gate metaclass as needed in order to target other platforms that provide new primitive operations.
- Regarding the intermediate level of the proposed approach, we could design a new metamodel that supports other models of quantum computation, or to consider hybrid computation (a program that comprises parts that will be executed on a classical computer and on a quantum one). In any case, this improvement will require, provided that no additional steps/metamodels are added to the toolchain, a new model-to-text transformation to generate the quantum code.
- Regarding the SAT problem, we could substitute the CNF metamodel by a more general one that allows users to model Boolean formulas in any form, not just CNF. This improved version could also provide other logical operators, such as XOR, implication, etc. In this case, a new model-to-model transformation will be needed to generate the appropriate quantum circuit, but the quantum metamodel and the last model-to-text transformation can be fully reused. Or, alternatively, we could develop a model-to-model transformation that generates a CNF model from an input Boolean formula, therefore reusing the developed infrastructure.
- Regarding the extension to tackle new problems, we could add new metamodels that provide the modelling concepts required by them, like factoring or quantum cryptography, reusing the implementation infrastructure already developed for generating Qiskit code.

The generation of executable quantum code from quantum circuits developed in this paper is quite straightforward. This is a direct consequence of having, in some way, both representations at the same abstraction level. The main contribution from the code part is adding the details in terms of the specific execution platform. It should be relatively easy to build reverse transformations to get the quantum circuits given the implementation in any quantum programming language, which enormously favour the portability between quantum platforms (assuming that some specific attributes or instructions specific of the platform would be added *ad hoc*).

The main difficulty in adopting an MDE approach for quantum program developments is on designing the transformation that generates the quantum circuit. In some way, this is not a surprise. Here is where we face the existing gap between modelling a problem from a non-quantum perspective and then obtaining the equivalent quantum one. As stated in [15], most of the current software problems are not specified in terms of probability spaces. Traditional software engineers make use of conceptual tools to model the system in the problem domain and, by means of transformations, get an executable representation where the concepts have found a direct correspondence. However, in quantum programming, the developer must deal with qubits and algebra operators to manipulate them, always having in mind the probability distribution of qubit values. In other words, there is a clear impedance mismatch between classical and quantum programming which will need of complex transformations.

Domain specific languages and patterns may be part of the solution. A quick view on the main literature on quantum computing is enough to see that most of the quantum programs are built around the use of some basic building quantum blocks or primitives. These include for instance the Grover diffuser, the QFT amplitude amplificator, the use of oracles, and some other operations for quantum arithmetic and logic. Indeed, there are some very well-known references [46] cataloguing the set of basic paradigms considered for implementing quantum programs. This means that, for a very representative set of problems to be solved, these primitives will be used, so it could be of interest to explore the conception of a quantum (domain) specific language which would serve to take those primitives and interconnect them in order to assemble a quantum solution as an aggregation of existing functionality. Still, it needs of much experience to exactly know what elements and in which order are needed to have a valid implementation. Certainly, the consideration of patterns to provide a set of common structures would be very helpful and it is a matter of interest for further research.

A good starting point would be having many examples of transformations. We believe that no one will dispute that the best demonstration of the usefulness of using MDE for quantum programs development will come through getting a big asset of problem-to-quantum circuit transformations. This is the best way to favour the definition of domain specific languages and patterns. We are only at the beginning of this task. Researchers have very recently started contributing to the definition of metamodels for the representation of quantum circuits. Our work is in this way a clear contribution as it is the first full implementation from the definition of the problem to the executable representation of the quantum circuit. The definition of the metamodel for representing quantum circuits, as described in this paper, is a reusable asset which will facilitate for sure the contribution of other researchers.

7. Conclusions and future work

In this paper we have demonstrated the suitability of the MDE approach for the automatic generation of quantum programs. A metamodel for representing quantum circuits is provided, which serves as an input model to generate Qiskit code, which is executable on real quantum computers and simulators. We conceived and implemented the approach to be extensible. Given the way in which quantum circuits are represented as models, other transformations to generate code to reach other platforms and languages are feasible.

The type of applications that can benefit from the proposed approach includes (i) those problems that have a well-known and structured quantum solution, which can be automatically generated (as has been the case for SAT, other problems, such as factorization of prime numbers or graph colouring, among others, fall in this category); and (ii) those applications that are implemented using some of the basic algorithms of QC and in which the programmer would have to manually complete the generated code (for instance, a quantum program for cryptography that generates the skeleton of the quantum circuit where the programmer has to add the part of the circuit that computes the information to be encrypted). As the number of implemented case studies grows, we also envision the scenario in which quantum code generation will be approached from a perspective similar to the one that has been followed in classical computing using design patterns, reference architectures and frameworks.

The case study adopted in this paper, the SAT problem, is very well-known and employed in the context of many engineering problems, like the validation of electronic circuits, for instance, to check the correctness of hardware designs. The gains in terms of reduced computation time provided by quantum solutions can improve not only the validation time of Electronic Design Automation software, but also enable it to carry out validations that are not currently possible given its complexity.

Although there are several notations to graphically represent quantum algorithms, the metamodeling contributes with a notation that is not only understood by quantum software engineers, but also it is a useful asset from which to extend existing tools. One of these possibilities is reverse SE, allowing developers to automatically generate UML-compliant models from existing quantum circuits, opening in this way even more the possibilities for integrating other validation and code generation tools (for instance, for getting a compiler to transform quantum programs between different execution platforms). Thus, the approach has two main benefits, apart from providing the automatic generation of quantum code: the possibility of integrating other existing SE tools, and the integration with existing UML models, of great interest for the implementation of hybrid systems (i.e., quantum exchanging data with classic computers).

The decision of splitting quantum circuits in slices has proven to be very useful when traversing the models to apply the transformation rules. Moreover, the solution adopted is not affected by the possible variations on the graphical representations of the quantum gates. Also, there is no limitation on the number of gates that can be added as they are only subclasses of the existing Gate superclass. The main result of this work is that executable quantum programs can be automatically obtained from other models at higher abstraction levels (that is, at the problem domain).

In terms of scalability of the length of the SAT equations that can be solved, the proposed approach is limited only by the characteristics of the available quantum infrastructure in terms of the number of qubits, and the maximum length of the quantum circuit (the longer the circuit, the higher the probability of de-coherence of the quantum system).

As a further extension we consider some refining steps in the transformation from the SAT problem to the quantum circuit to reduce the number of gates and qubits required. Furthermore, the transformation step from the quantum model to the Qiskit code can be extended with a set of constraints to assure that the origin model is well-formed according to the quantum restrictions and the limitations of the chosen infrastructure.

This paper is, to our knowledge, the first demonstration of a full transformation, using MDE techniques, from a problem to be solved (expressed as a model conformed to a given metamodel) to executable quantum code. As this, long-term research is ahead of us in order to facilitate the development of quantum programs and its integration within the community of classical software developers. But it is nevertheless a need given the huge conceptual leap that exists between the concepts employed by both disciplines.

CRediT authorship contribution statement

Diego Alonso: Conceptualization, Software, Validation, Investigation, Writing – original draft, Writing – review & editing. **Pedro Sánchez:** Conceptualization, Methodology, Investigation, Writing – original draft, Writing – review & editing. **Francisco Sánchez-Rubio:** Software, Writing – review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We acknowledge the use of IBM Quantum services for this work. The views expressed in this paper are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team.

References

[1] Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer (revised version of the original paper, published in 1994). SIAM Rev 1999;41(2):303–32. https://doi.org/10.1137/S0036144598347011.

[2] Grover L. K.. A fast quantum mechanical algorithm for database search, in: Proceedings of the 28th annual ACM symposium on theory of computing. STOC '96, Association for Computing Machinery, New York, NY, USA1996;:212–219. 10.1145/237814.237866.

[3] Piattini M, Serrano M, Perez-Castillo R, Petersen G, Hevia JL. Toward a quantum software engineering. IT Prof 2021;23(1):62–6. https://doi.org/10.1109/MITP.2020.3019522.

[4] Mohseni M, Read P, Neven H, Boixo S, Denchev V, Babbush R, Fowler A, Smelyanskiy V, Martinis J. Commercialize quantum technologies in five years. Nature 2017;543(7644):171–4. https://doi.org/10.1038/543171a.

[5] Piattini M, Peterssen G, Pérez-Castillo R. Quantum computing: a new software engineering golden age. SIGSOFT Softw Eng Notes 2020;45(3):12–4. https://doi.org/10.1145/3402127.3402131.

[6] Selic B. The pragmatics of model-driven development. IEEE Softw 2003;20(5):19–25. https://doi.org/10.1109/MS.2003.1231146.

[7] Knuth D. The art of computer programming. Satisfiability. vol. 4. Addison–Wesley; 2016.

[8] Biere A, Heule M, Maaren HV, Walsh T. Handbook of satisfiability (frontiers in artificial intelligence and applications). IOS Press; 2009.

[9] Cook S.A.. The complexity of theorem-proving procedures, in: Proceedings of the third annual ACM symposium on theory of computing. STOC '71, Association for Computing Machinery, New York, NY, USA1971;:151–158. 10.1145/800157.805047.

[10] Barenco A, Bennett CH, Cleve R, DiVincenzo DP, Margolus N, Shor P, Sleator T, Smolin JA, Weinfurter H. Elementary gates for quantum computation. Phys Rev A 1995;52:3457–67. https://doi.org/10.1103/PhysRevA.52.3457.

[11] Heim B, Soeken M, Marshall S, Granade C, Roetteler M, Geller A, Troyer M, Svore K. Quantum programming languages. Nat Rev Phys 2020;2(12):709–22. https://doi.org/10.1038/s42254-020-00245-7.

[12] Piattini M., Paradela C.A., Phillipson F., Pérez-Castillo R., Guzmn I., Serrano M., Polo M., Gonzlez G.H., Oliver J.L.H., Marqueo J., Murina E., Nodarse G.P.. The talavera manifesto for quantum software engineering and programming. In: Proceedings of the international workshop on the QuANtum SoftWare engineering & programming. 2020b.

[13] Jianjun Zhao. 2021. Quantum Software Engineering: Landscapes and Horizons. arXiv:2007.07047 [cs.SE] (accessed Aug. 2022). arXiv:2007.07047.

[14] Dey N, Ghosh M, kundu SS, Chakrabarti A. QDLC - the quantum development life cycle. arXiv:2010.08053 (accessed Aug. 2022).

[15] Sánchez P, Alonso D. On the definition of quantum programming modules. Appl Sci 2020;11(13). https://doi.org/10.3390/app11135843.

[16] Ali S, Yue T. Modeling quantum programs: challenges, initial results, and research directions 2020:14–21. https://doi.org/10.1145/3412451.3428499.

[17] Bézivin J. On the unification power of models. Softw Syst Model 2005;4(2):171–88. https://doi.org/10.1007/s10270-005-0079-0.

[18] Atkinson C, Khne T. Model-driven development: a metamodeling foundation. IEEE Softw 2003;20(5):36–41. https://doi.org/10.1109/MS.2003.1231149.

[19] Selic B. The pragmatics of model-driven development. IEEE Softw 2003;20(5):19–25. https://doi.org/10.1109/MS.2003.1231146.

[20] Rodrigues A, Silva D. Model-driven engineering: a survey supported by the unified conceptual model. Comput Lang Syst Struct 2015;43:139–55. https://doi.org/10.1016/j.cl.2015.06.001.

[21] Kahani N, Bagherzadeh M, Cordy J, Dingel J, Varro D. Survey and classification of model transformation tools. Softw Syst Model 2020;18. https://doi.org/10.1007/s10270-018-0665-6.

[22] Sendall S, Kozaczynski W. Model transformation: the heart and soul of model-driven software development. IEEE Softw 2003;20(5):42–5. https://doi.org/10.1109/MS.2003.1231150.

[23] R. Pérez-Castillo, L. Jiménez-Navajas, M. Piattini. Modelling quantum circuits with uml. Proceedings - 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering, Q-SE 2021, 7-12. doi 10.1109/Q-SE52541.2021.00009.

[24] Gemeinhardt F., Garmendia A., Wimmer M.. Towards model-driven quantum software engineering. Proceedings of the 2nd international workshop on quantum software engineering, co-located with ICSE2021;. 10.5281/zenodo.4593888.

[25] Moin A, Challenger M, Badii A, Gnnemann S. Mde4qai: towards model-driven engineering for quantum artificial intelligence. arXiv:2107.06708 (accessed Aug. 2022).

[26] Barrett C., Sebastiani R., Seshia S., Tinelli C.. Frontiers in artificial intelligence and applicationsCh Satisfiability Modulo Theories, IOS Press2009; 185:825–885. 10.3233/978-1-58603-929-5-825.

[27] Johnston ER, Harrigan N, Gimeno-Segovia M. Programming quantum computers: essential algorithms and code samples. O'Reilly Media; 2019.

[28] Sutor R. Dancing with qubits: how quantum computing works and how it can change the world. Packt Publishing; 2019.

[29] Ambainis A. Quantum search algorithms. SIGACT News 2004;35(2):22–35. https://doi.org/10.1145/992287.992296.

[30] Cheng S-T, Tao MH. Quantum cooperative search algorithm for 3-sat. J Comput Syst Sci 2007;73(1):123–36. https://doi.org/10.1016/j.jcss.2006.09.003.

[31] Wang P., Liu G., Liu L.. A generic variable inputs quantum algorithm for 3-sat problem. Proceedings of the IEEE international conference on advances in electrical engineering and computer applications(AEECA)2020;:308–312. 10.1109/AEECA49918.2020.9213471.

[32] Schoning T.. A probabilistic algorithm for k-sat and constraint satisfaction problems. Proceedings of the annual symposium on foundations of computer science1999;:410–414. 10.1109/SFFCS.1999.814612.

[33] Perdomo-Ortiz A, Venegas-Andraca SE, Aspuru-Guzik A. A study of heuristic guesses for adiabatic quantum computation. Quantum Inf Process 2011;10(1):33–52. https://doi.org/10.1007/s11128-010-0168-z.

[34] Campos E, Venegas-Andraca SE, Lanzagorta M. Quantum tunneling and quantum walks as algorithmic resources to solve hard k-sat instances. Sci Rep 2021;11(1):16845. https://doi.org/10.1038/s41598-021-95801-1.

[35] Many Qiskit: an open-source framework for quantum computing, 2021. https://raw.githubusercontent.com/Qiskit/qiskit/master/Qiskit.bib.

[36] Kolovos D, Paige R, Polack F. The epsilon transformation language. Proceedings of the theory and practice of model transformations conference, ICMT. 5063, Springer; 2008. https://doi.org/10.1007/978-3-540-69927-9_4.Lecture Notes in Computer Science, pp. 46–60
[37] Boyer M, Brassard G, Hayer P, Tapp A. Tight bounds on quantum searching. Fortschr Phys 1998;46(4–5):493–505. https://doi.org/10.1002/(SICI)1521-3978(199806)46:4/5.
[38] Nielsen MA, Chuang IL. Quantum computation and quantum information. Cambridge University Press; 2000.
[39] Qcore repository, https://github.com/DiegoAlonso/sat_qcore (accessed Aug. 2022).
[40] Kahani N, Bagherzadeh M, Cordy JR, Dingel J, Varró D. Survey and classification of model transformation tools. Softw Syst Model 2019;18(4):2361–97. https://doi.org/10.1007/s10270-018-0665-6.
[41] Baudry B, Ghosh S, Fleurey F, France R, Traon YL, Mottu JM. Barriers to systematic model transformation testing. Commun ACM 2010;53(6):139–43. https://doi.org/10.1145/1743546.1743583.
[42] Ciancone A., Filieri A., Mirandola R.. Mantra: towards model transformation testing. Proceedings of the 7th international conference on the quality of information and communications technology2010;:97–105. 10.1109/QUATIC.2010.15.
[43] Aranega V, Mottu J-M, Etien A, Degueule T, Baudry B, Dekeyser JL. Towards an automation of the mutation analysis dedicated to model transformation. Softw Test Verif Reliab 2015;25(5–7):653–83. https://doi.org/10.1002/stvr.1532.
[44] Mottu J.-M., Sen S., Tisi M., Cabot J.. Static analysis of model transformations for effective test generation. Proceedings of the IEEE 23rd international symposium on software reliability engineering2012;:291–300. 10.1109/ISSRE.2012.7.
[45] A. Younes. Strength and weakness in Grover's quantum search algorithm. (Aug. 2022) arXiv:0811.4481 [quant-ph].
[46] Many authors. Quantum Algorithm Implementations for Beginners. ACM Transactions on Quantum Computing 3, 4, Article 18 (December 2022). https://doi.org/10.1145/3517340.