ARCHIWUM INFORMATYKI
TEORETYCZNEJ I STOSOWANEJ
Tom 17 (2005), z. 4
pp 265-272

Description and visualisation of quantum circuits with XML

Jarosław A. Miszczak[a]

[a]Institute of Theoretical and Applied Informatics
Polish Academy of Sciences
Bałtycka 5, 44-100 Gliwice, Poland
email: `miszczak@iitis.gliwice.pl`

Abstract: This paper describes software for transformation of XML-based description of quantum circuits into graphical representation. It uses Quantum Markup Language introduced in [8] and produces Scalable Vector Graphics (SVG) representation of data.

Because XML is a popular data format supported by many programming languages, presented converter does not depend on simulation environment and can be easily connected with existing software for simulation of quantum computing.

Problems with visualisation of quantum circuit description point out some limitations of quantum circuits model in practical application *e.g.* quantum programing languages.

1. Introduction

Introduced in late nineties XML became *de facto* standard for representing and interchanging data. Abundance of tools based on it allows to process XML documents on any hardware and software platform.

Recently few XML based file formats for interchanging and storing information on quantum circuits were proposed [10, 8]. In first example XML is used as a language of commands for the simulator. In the second work Quantum Markup Language was introduced for interchanging data between simulation engine and web based user interface. Also software from others fields of science utilises XML for storing information about various physical objects [3] or data required for computation or simulation [4].

This paper describes module, written in Python [1] programming language, which converts Quantum Markup Language description of quantum circuit into graphical representation using Scalable Vector Graphics [9]. Python was chosen because of easy integration with existing infrastructure of Fraunhofer Quantum Computing Simulator.

This paper is organised as follows. First we present basic elements of Quantum Markup Language[1] and discuss advantages and disadvantages of this language. Next we briefly present method of transformation used in described software and include some examples. We conclude pointing out limitations of the quantum circuit model as a language for description of the quantum information processing.

2. Platform independent notation for quantum circuits

Quantum circuits model [6] is one of the most popular among scientists interested in quantum information theory. It is mainly used for representation of unitary gates but it also to some extend allows to represent arbitrary quantum computational process, including measurement.

As we will see this model does not allow to express some elements useful in simulations of quantum computers. In that sense described software shows limitations of the quantum circuits model as a quantum programming technique.

2.1. Quantum gates as markups

Extensible markup languages can be used for representation of quantum gates in a very convenient way. Here we present basic elements of Quantum Markup Language [8].

Quantum Markup Language (QML) document contains `<QML>` element, which is a parent for five elements, namely: `<Job>`, `<Circuit>`, `<GateLib>`, `<CircuitLib>` and `<Results>`

Listing 1 presents an example of QML document. Its graphical representation is presented in Fig. 1.

Only `<Circuit>` element is required for definition of quantum circuit. It contains the description of quantum circuits and it contains `<Operation>` elements, which represent steps of the quantum algorithm. Attribute `Size` of tag `<Circuit>` defines number of qubits required for execution of the presented circuit.

Each `<Operation>` defines – using attribute `bits` of `<Application>` element – bits on which gates should be performed. It is natural since information required for proper execution of gates is independent from their location. This can be used to include parts of external descriptions of circuit into QML document and is similar to mechanism of functions in procedural programming languages.

[1]Complete reference with description of all gates can be found in [8].

Since every popular programming language include support for XML it is easy to add support for QML to any existing software for simulation of quantum computers and to integrate it with existing simulation platform.

One of the disadvantages of QML is the lack of precise specification which prevents validation of documents.

```
<?xml version="1.0"?>
<QML>
  <Circuit Name="test1"
    Id="simple.qml" Size="4" >
    <Operation Step="0">
    </Operation>
    <Operation Step="1">
      <Application Name="G" Bits="0,1">
        <Gate Type="CNOT"/>
      </Application>
      <Application Name="G" Bits="2">
        <Gate Type="HADAMARD"/>
      </Application>
      <Application Name="G" Bits="3">
        <Gate Type="HADAMARD"/>
      </Application>
  </Operation>
  <Operation Step="2">
      <Application Name="G" Bits="2,3">
        <Gate Type="CNOT"/>
      </Application>
  </Operation>
  <Operation Step='3'>
    <Application Bits='1'>
      <Gate Type="PHASE" Divisions="2"/>
    </Application>
  </Operation>
    <Operation Step='4'>
    <Application Bits='1,2,3'>
      <Gate Type="TOFFOLI"/>
    </Application>
  </Operation>
  </Circuit>
</QML>
```

Listing 1: Example of a QML document. This document contains only `<Circuit>` section which defines sequence of gates executed by simulation engine. Also every gate presented here can be directly implemented by unitary operation. This is not always true because QML allows for conditional execution of gates.

2.2. External circuits

Fig. 1: Simple circuit with controlled gates. Controlled qubits are marked with filled circles. SVG documents can use CSS for specification of many attributes of graphical elements like colour or line thickness. More examples can be found in [5].

Fig. 2: Circuit containing calls to other circuits. Group of gates in frame is included in circuit using `<Circuit>` element, which is a child of `<Operation>` element.

```
<?xml version="1.0"?>
<QML>
<Circuit Name="external" Id="external.qml" Size="5">
  <Operation Step="0">
  </Operation>
  <!-- ... -->
  <Operation Step='4'>
    <Application Bits='1,2,3'>
      <Gate Type="CIRCUIT"
      href="http://path/to/file.qml"/>
    </Application>
  </Operation>
  <!-- ... -->
</Circuit>
</QML>
```

Listing 2: QML document can contain `<Circuit>` gates with link to external definition of gate. Such construction can be used to build library of circuits and connect them dynamically during execution. Attribute `href` of the tag `<Circuit>` contains URL of the file with the definition of the circuit.

One of the most interesting features of QML is the possibility of including external descriptions of circuits. This allows to prepare parts of simulation using different tools and connect them using common XML format. This also allows use simulation engine of the FHG simulator [8] with tools such QCL [7], providing conversion of their output to QML.

Visual representation of circuit (see: Listing 2) containing reference to external circuit is presented in Fig. 2.

External elements of circuit are included using `<Circuit>` element using its `href` attribute as it is presented in Listing 2. Since most of the programming languages support popular Internet protocols (*e.g.* HTTP org FTP), it is easy to divide generation of resulting QML document among many remote hosts.

2.3. QML processing in Python

Python [1] is a very popular, general purpose programming language. It is platform independent and provides rich set of module for processing XML documents [2].

The conversion of QML description is straightforward in Python. Package `qml2svg` contains classes `qml.Circuit` and `qml.Gate` for internal representation of circuit and its elements and class `qml.svgGate`, which contains methods responsible for visualisation of gates. Class `qml.svgGate` contains the code specific for gates used in Fraunhofer Quantum Computer Simulator, but new types of gates can be added easily.

Process of conversion is carried out by `qml.qmlConverter` class, which stores information about output format – SVG is default and simple text format, useful for debugging purpose, is also supported. Method `parseQML` of the class `qml.qmlConverter` reads data stream and produces internal representation of circuit. The method `showCircuit` using this representation calls appropriate methods for gates in circuit and produces an output document.

```
import qml
import liburl

# open and read remote resource
file = open(options.qml)
qml = file.read()

# parameter of defines output format
converter = qml.qmlConverter(of="svg")

print converter.showCircuit(qml)
```

Class `qml.Gate` is used as a container for information about parameters of gate (they are stored in `params` list) and contains the method `show`, which calls appropriate methods in `qml.svgGate`.

Beacause gates' tags do not contain information about qubits on which gates are supposed to be executed class `qml.Gate` uses information from `qml.Circuit` while calling methods responsible for drawing specific gates. Some gate must have defined specific parameters which are necessary for visualisation, *e.g.* gate `<CPHASE>` must have parameter `Division` in its `params` list. There is only one object of class `qml.Circuit` needed for representation of `<Circuit>` and it contains the list of executed gates and information about their target.

Gate `<CIRCUIT>`[2] has attribute `href` which contains URL of the file with the definition of the circuit. Since Python provides `liburl` standard module which allows to use remote resources similarly like local files, it easy to process information about quantum circuit contained remote file. Reference to external circuit is used to read data and create SVG representation for every gate, but it does not create full SVG document. Gates form external circuit are appended to output document as a new group of graphical elements.

3. Conditional execution

QML contains not only description of quantum gates but also information for simulation engine. This information is expressed by markups which are problem-

[2]Note that this gate is a child of the tag `<Circuit>`

atic for visualisation. Example of them is `<Random>` gate, which produces random unitary transformation or usage of `CaseIndeces` arttribute in `<CIRCUIT>` gate (see: Listing 3), which allows to include conditionally external circuits.

```
<?xml version="1.0"?>
<QML>
<Circuit Name="cond" Id="cond.qml" Size="5">
  <Operation Step="0">
  </Operation>
  <!-- some operations. -->
  <Operation Step='4'>
    <Application Bits='1,2,3'>
      <Gate Type="CIRCUIT" Size="5" CaseIndices="0,1"
      href0="http://qc.fhg.de/clib/c0.qml"
      href1="http://qc.fhg.de/clib/c1.qml"
      href2="http://qc.fhg.de/clib/c2.qml"
      href3="http://qc.fhg.de/clib/c3.qml"/>
    </Application>
  </Operation>
</Circuit>
</QML>
```

Listing 3: Example of circuit with conditional execution of gates. `CaseIndeces` defines on which qubits should be performed measurement in order to obtain classical bits. Next different subcircuits defined by URLs `href0`, `href1`, . . . , `hrefN` are called to according to the values of those bits.

Though this elements are trivial for realisation by simulation engine, they show that quantum computational model described in XML allows to express some elements better than quantum circuit model, based only on unitary transformations. Thus there is a need for language which allows to incorporate control structures more complicated than those commonly used in standard quantum circuits model.

4. Final remarks

We have presented method of description and visualisation of quantum circuit model based on XML. We have shown that Quantum Markup Language is powerfull enough to controll quantum machines better then quantum circuits. It provides many features which should be implemented in quantum programming language to get precise control over process of quantum computation.

The most interesing features are those using probabilistic execution. Since quantum computing is probabilistic those elements should be implemented in the high level quantum programming language.

Acknowledgements

This paper has been supported by the Polish Ministry of Scientific Research and Information Technology under the (solicited) grant No PBZ-MIN-008/P03/2003.

Author would like to thank Dr. Helge Rosé and Fraunhofer Quantum Computing team for cooperation.

References

[1] Python programming language home page. http://www.python.org/.

[2] *Python/XML Libraries.* http://pyxml.sourceforge.net/.

[3] D. Binosi and L. Theußl. JaxoDraw: A graphical user interface for drawing Feynman diagrams. *Computer Physics Communications*, 161, 2004.

[4] G. Collecutt and P. D. Drummond. xmds: eXtensible multi-dimensional simulator. *Computer Physics Communications*, 142, 2001.

[5] J. A. Miszczak. Converter qml2svg. Web page with software and examples: http://www.iitis.gliwice.pl/zksi/stuff/qml2svg/.

[6] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.

[7] B. Ömer. Quantum programming in QCL. Master's thesis, TU Vienna, 2000.

[8] H. Rosé, T. Asselmeyer-Maluga, M. Kolbe, F. Niehoerster, and A. Schramm. The fraunhofer quantum computing portal – www.qc.fraunhofer.de – a web-based simulator of quantum computing processes. http://www.arxiv.org/abs/quant-ph/0406089, 2004.

[9] Scalable vector graphics (SVG) 1.1 specification. W3C recommendtaion, W3C, 2003. http://www.w3.org/TR/SVG11/.

[10] P. Wocjan, M. Eck, and R. M. Zeier. Quasi – Quantum Circuit Simulator. Technical report. http://iaks-www.ira.uka.de/QIV/QuaSi/.

Opis i wizualizacja obwodów kwantowych z wykorzystaniem XML

Streszczenie

Artykuł omawia oprogramowanie suce do transformacji opartego na XML-u formatu opisu obwodów kwantowych do reprezentacji graficznej. Oprogramowanie to bazuje na języku Qunantum Marku Language, stworzonego w ramach prjektu Fraunhofer Quantum Computing Simulator i wykorzystuje format SVG do reprezentacji graficznej obwodu.

Poniewa XML jest popularnym formatem danych wspieranym przez wiele języków programowania, prezentowany konwerter nie zależy od konkretnego symulatora i może być połączony z istniejącym oprogramowaniem służącym do symulacji obliczeń kwantowych.

Jednocześnie problemy z wizualizacją obwodów kwantowych wskazują na ograniczenia modelu obwodów kwantowych w praktycznych zastosowaniach takich jak kwantowe języki programowania.