Wireframe Model for Simulating Quantum Information Processing Systems

Mizal Alobaidi[1], Andriy Batyiv[2], and Grygoriy Zholtkevych[2]

[1] Tikrit University,
Faculty of Computer Science and Mathematics, P.O. Box–42, Tikrit, Iraq
mizalobaidi@yahoo.com
[2] V.N. Karazin Kharkiv National University,
School of Mathematics and Mechanics, 4, Svobody Sqr., 61022, Kharkiv, Ukraine
{generatorglukoff,g.zholtkevych}@gmail.com

Abstract. This paper continues the papers series concerned with authors' research of quantum information processing systems based on the formal model of abstract quantum automata. The previous papers of the series were focused on mathematical modelling of quantum information processing systems. This paper describes the core components of information technology for studying the systems by computer simulation. A domain model and a behavioural model of the wireframe for simulating quantum information processing system are presented in the paper. The language "AQuanAut" for description abstract quantum automata is specified. The problems encountered during the realizing of the wireframe model are discussed.

Keywords. Simulation, simulation wireframe, domain model, behavioural model, quantum computation, finite-level quantum system, quantum information processing system, abstract quantum automaton

Key terms. MathematicalModel, ComputerSimulation, Specification-Process

1 Introduction

Necessity to increase processing power for computational devices, traffic capacity and security level for communication channels leads to new challenges in the fields of Information and Communication Technology (ICT). Nowadays quantum informatics is considered as an approach to meet the challenges. The research in the area of quantum informatics uses knowledge in the fields of mathematics, physics, and computer science. So, the corresponding research technique covers the wide variety of methods including both theoretical developments and experimental investigations. We should note that quantum experiments are quite difficult and expensive for accomplishing. Therefore, the problem to simulate such experiments arises naturally.

This paper describes an attempt to construct the wireframe model for simulating quantum information processing systems basing on the notion of an abstract quantum automaton [1, 2]. Taking in account that majority of specialists in computer science are not familiar with the notation and concepts of quantum informatics in detail authors have tried to provide maximal presentation completeness for the notation and basic concepts. As needed, one can use [5] for more deep acquaintance with problems and methods of quantum informatics.

2 Brief Survey of Physical Grounds and Mathematical Models

In the section the description of mathematical model for quantum information processing systems in the terms of abstract quantum automata and physical grounds for the model are given.

At the highest abstraction level an abstract quantum automaton can be considered as a two-component hybrid quantum-classical system $\mathfrak{A}(\mathcal{Q}, \mathcal{T})$. The quantum component $\mathcal{Q}$ of the system functions as a memory and the classical component $\mathcal{T}$ of the system implements an information process control. Interactions between the classical control component and the quantum memory sustain the integrity of the system.

Now let's suppose satisfiability a number of properties for systems $\mathcal{Q}$ and $\mathcal{T}$.

Assumption 1 *The quantum memory $\mathcal{Q}$ is an m-level quantum system.*

It means that an m-dimensional Hilbert space $\mathcal{H}_m$ is associated with the quantum memory. This space is known as the state space. The memory is completely described by its pure state, which is a one-dimensional subspace of the state space. This subspace is uniquely represented by the ortho-projector $|\psi\rangle\langle\psi|$ on the unit vector $|\psi\rangle$ which generates the subspace.

In contrast to pure states mixed states are used to describe memory whose state is not completely known. Rather more detailed suppose we know that a memory is in one of a number of states $\{|\psi_k\rangle\langle\psi_k| : k = 1, \ldots, s\}$ with respective probabilities $\{p_k\colon k = 1, \ldots, s\}$. We shall call $\{p_k, |\psi_k\rangle\langle\psi_k| : k = 1, \ldots, s\}$ an ensemble of pure states. The density operator for the system is defined by the equation $\rho = \sum_{k=1}^{s} p_k |\psi_k\rangle\langle\psi_k|$.

Mixed states are identified with density operators. The statement that pure states are described by one-dimensional ortho-projectors allows to consider pure states as indecomposable states.

Assumption 2 *The control system $\mathcal{T}$ is a deterministic labelled transition system.*

It means that $\mathcal{T}$ is a tuple $(C, n_*, T, \Lambda, \sharp, Trans, \mathrm{dom}, \mathrm{codom}, \lambda)$ where

- C is a finite set of computational nodes;
- n_* is some element of C, which is called the initial node;
- T is a finite set of terminal nodes such that $C \cap T = \emptyset$;

- Λ is a finite alphabet of possible outcomes;
- $\sharp$ is some picked outcome in Λ;
- $Trans$ is a finite set of transitions;
- $\mathrm{dom} : Trans \to C$ maps each transition into the node, which is source of this transition;
- $\mathrm{codom} : Trans \to N$ maps each transition into the node, which is sink of this transition, where $N = C \cup T$;
- λ is a map from $Trans$ onto Λ.

The following conditions should be satisfied for this tuple

determinacy: for any $\tau', \tau'' \in Trans$ equalities $\mathrm{dom}(\tau') = \mathrm{dom}(\tau'')$ and $\lambda(\tau') = \lambda(\tau'')$ imply $\tau' = \tau''$;

default label: for any $\tau \in Trans$ the equality $\lambda(\tau) = \sharp$ is equivalent to $\mathrm{dom}^{-1}(\mathrm{dom}(\tau)) = \{\tau\}$;

reachability: for any $n \in N$ there exists a finite sequence $\tau_1, \ldots, \tau_k \in Trans$ such that $\mathrm{dom}(\tau_1) = n_*$, $\mathrm{codom}(\tau_k) = n$, and for all $j = 1, \ldots, k-1$ the following equality $\mathrm{codom}(\tau_j) = \mathrm{dom}(\tau_{j+1})$ is true.

Assumption 3 *A snapshot of an abstract quantum automaton is completely described by the pair $|\psi\rangle, n$, where $|\psi\rangle \in \mathcal{H}_m$ is a unit vector represented the current memory state and $n \in N$ is some node of the control system.*

Assumption 4 *Each interaction between the memory and the control system is a pair consisting of a memory state transformation and a jump from one node to another: $|\psi\rangle, n \vdash |\psi'\rangle, n'$. This jump should be determined by a transition: if $|\psi\rangle, n \vdash |\psi'\rangle, n'$ then there exists the unique transition $\tau \in Trans$ realizing the jump, i.e. $\mathrm{dom}(\tau) = n$ and $\mathrm{codom}(\tau) = n'$. Such interactions are called quantum actions.*

Assump. 4 does not determine any algorithm for performing quantum actions. The next assumption describes explicitly such an algorithm for removing this defect. This description uses the notion of a generating isometric operator for quantum actions. The notion has introduced and studied in detail in [1].

Assumption 5 *The quantum action associated with a computational node n is described by its generating isometric operator $W_n : \mathcal{H}_m \otimes l^2(Out_n)$, where $Out_n = \lambda(\mathrm{dom}^{-1}(n))$. This operator determines the interaction $|\psi\rangle, n \vdash |\psi'\rangle, n'$ by using the following procedure:*

1. *select randomly the label $a \in Out_n$ in accordance with the probability distribution*
$$\Pr(a \mid \psi) = \langle\psi|W_n^\dagger(\mathbf{1} \otimes |a\rangle\langle a|)W_n|\psi\rangle, \tag{1}$$
where $W_n^\dagger$ is the adjoint operator to the operator W_n, $|a\rangle(\cdot) = \delta(a, \cdot)$, and $\delta(\cdot, \cdot)$ is Kronecker delta;
2. *determine the transition $\tau \in Trans$ such that $\lambda(\tau) = a$ and $\mathrm{dom}(\tau) = n$;*

3. compute the pair $|\psi'\rangle, n'$ *by the following formulae*

$$|\psi'\rangle = \frac{J(a)^\dagger W_n |\psi\rangle}{\sqrt{\Pr(a \mid \psi)}} \tag{2}$$

$$n' = \mathrm{codom}(\tau) \tag{3}$$

where the isometric operator $J(a)$ *acts from* $\mathcal{H}_m$ *into* $\mathcal{H}_m \otimes l^2(\Lambda)$ *in compliance with the next formula* $J(a)|\phi\rangle = |\phi\rangle \otimes |a\rangle$.

Thus, a trajectory of an abstract quantum automaton can be define as a sequence of interactions $|\psi_0\rangle, n_* \vdash |\psi_1\rangle, n_1 \vdash \cdots \vdash |\psi_t\rangle, n_t$, where n_t is a terminal node. The corresponding sequence of labels $\lambda(\tau_1)\lambda(\tau_2)\ldots\lambda(\tau_{t-1})$ such that $\mathrm{dom}(\tau_1) = n_*$ and $\mathrm{dom}(\tau_j) = n_j$, $\mathrm{codom}(\tau_j) = n_{j+1}$, where $j = 1, \ldots, t-1$, will be called an automaton trace. Stress that only an automaton trace is an observed part of the automaton trajectory.

3 Description of Simulation Wireframe Model

In the paper the Unified Modelling Language (UML) [6, 7] is used for specifying different details of quantum information processing systems. Object Constraint Language (OCL) expressions [3] are added to UML diagrams to describe a model precisely.

Describing the simulation wireframe model of abstract quantum automata we start with a specification of their composite structure (see Fig. 1). The component `memory` describes the quantum memory (the finite-level quantum system $\mathcal{Q}$) and the component `control` is a model for the classical control system $\mathcal{T}$.

The structural units of the component `control` describe three kinds of its constituents: nodes (N), transitions ($Trans$), and labels (Λ).

More ample description of an abstract quantum automaton structure is described by domain model (see Fig. 2). As it is shown in Fig. 2 the set of nodes is divided into two subsets of nodes: the subset of computational nodes and the subset of terminal nodes.

Computational nodes have two differences from the terminal nodes. Firstly, some quantum action is associated with each computational node in contrast to a terminal node. Secondly, a computational node has as minima one outgoing transition, whereas each terminal node has not any outgoing transition.

Properties of the control system for an abstract quantum automaton is set by Assump. 2. The static instance `default` of the class `Label` encapsulated into this class represents the specific label (so called the default label) for transitions that are determined uniquely by their sources and sinks. Existence of the default label is grounded by the "default label" condition. The determinacy condition of the control system can be described by the next constraint

```
context Transition inv: determinacy
   Transition::allInstances()->forAll(t1, t2: Transition |
      t1.dom = t2.dom and t1.tag = t2.tag implies
      t1.codom = t2.codom)
```

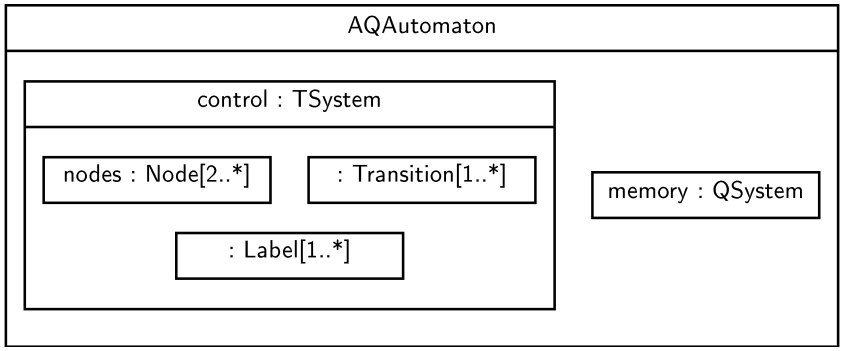

Fig. 1. Composite structure of an abstract quantum automaton

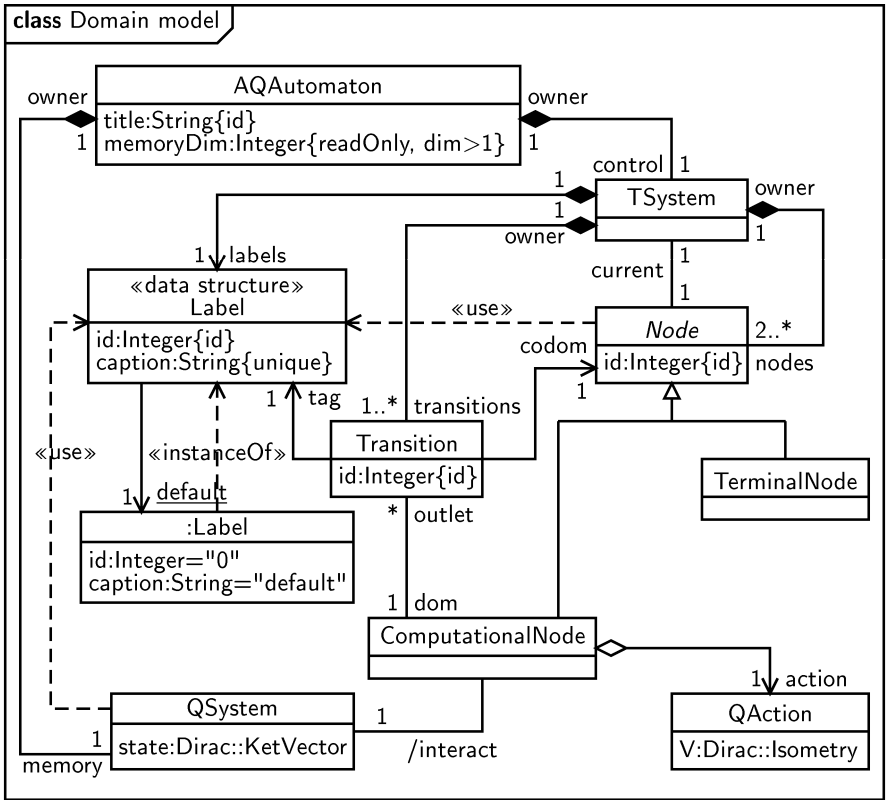

Fig. 2. Abstract quantum automaton domain model

Sustaining an interaction between the memory and the control system requires visibility of the component `memory` from the current computational node of the control system and vice versa. The derived association `interact` is intended to solve this task.

```
-- The definition of the association 'interact'
context ComputationalNode -- view from the control system
   def: interact: QSystem =
           self.current.owner.memory
context QSystem -- view from the memory
   def: interact: ComputationalNode =
           self.owner.control.current
```

The generalized model of an abstract quantum automaton behaviour is presented in Fig. 3.

This model specifies the behaviour of an abstract quantum automaton in compliance with [2, Def. 11]. The corresponding mathematical representation of all dynamical aspects for the interaction of automaton components are collected in Assump. 5.

The solution to use two concurrent threads for realizing the evolution of an automaton makes possibility to interrupt a simulation process. Necessity of the possibility has been established in the course of testing a trial implementation of the wireframe.

Initialization of an automaton is realized by the methods `TSystem::reset()` and `QSystem::set(state:Dirac::KetVector)`.

The method `QSystem::step(action: QAction)` implements a quantum character of the behaviour for the automaton. Detailed specification of the method is shown in Fig. 4. Each performance of this methods leads to changing the current computational node. Directly, changing of the current computational node is effected by the method `TSystem::setCurrent(outcome:Label)` specified by the next constraint

```
-- The rule for changing the current node
context TSystem::setCurrent(outcome:Label)
post: control =
         control@pre.transitions->any(tag = outcome).codom
```

This rule corresponds to Assump. 4 and formula (3) of Assump. 5.

The interaction shown in Fig. 4 implements one automaton jump (see Assump. 4).

Two methods of the class `QSystem` (`selectOutcomes(action:QAction)` and `getDistr(outs:Label[1..*],action:QAction)`) are used for building the probability distribution associated with the current memory state and the current action. Formula (1) of Assump. 5 is used to calculate this distribution.

Selection of one of the possible outcomes is effected by using the standard generator of random real numbers uniformly distributed in the segment $[0, 1]$.

Finally, formula (2) of Assump. 5 is applied to calculate new memory state under condition that outcome of the action is known.

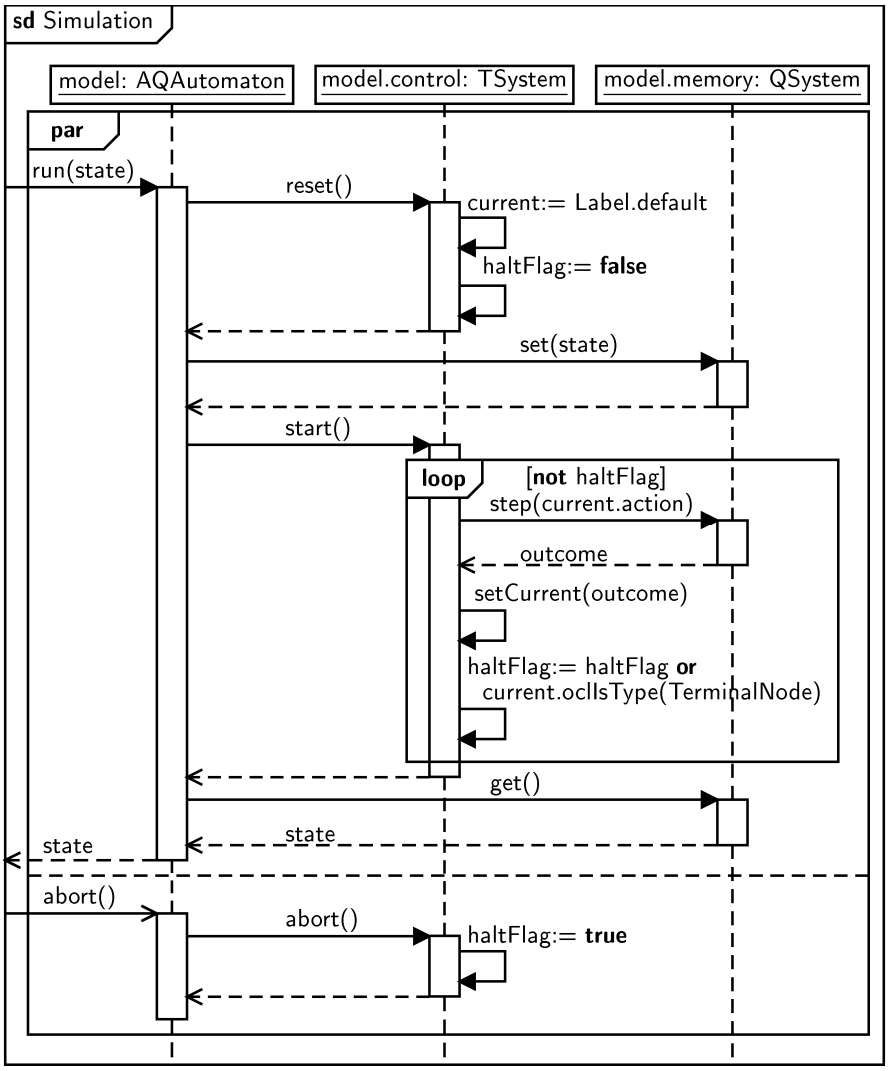

Fig. 3. Abstract quantum automaton behavioural model: interaction of components

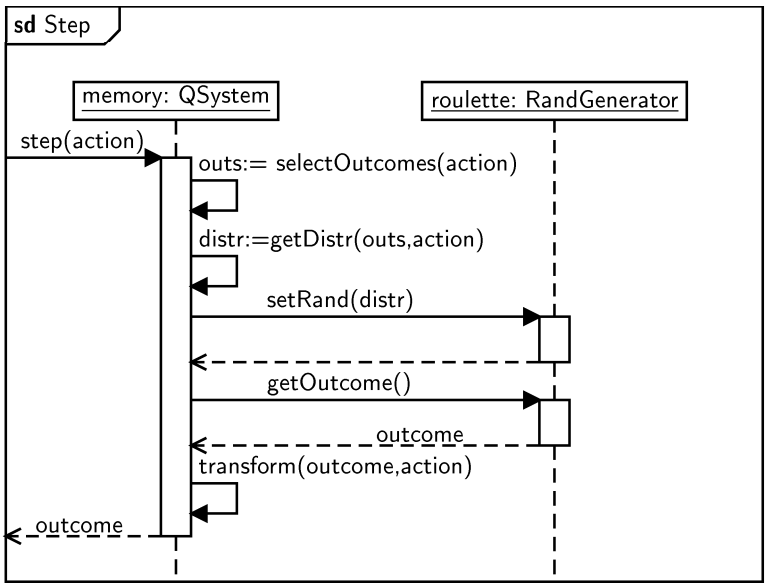

Fig. 4. Abstract quantum automaton behavioural model: step interaction

The label `outcome` returned by the method `step(QAction:action)` is used for changing the current node as it was discussed above.

4 Language "AQuanAut" for Specifying Abstract Quantum Automata

Language "AQuanAut" is a specification language for describing abstract quantum automata. Each AQuanAut-specification presents the description for the corresponding abstract quantum automaton. This description should be provide sufficient data to build the programmed simulator for investigating the described automaton. For example, authors used these description as input data for Java-application, which is a builder of abstract quantum automaton Java-simulators.

Syntax of language "AQuanAut" is below presented by use of Extended Backus - Naur Form [4].

The start symbol of "AQuanAut" grammar is denoted by `specification` in the paper. It is defined by the next rule, which fixes two-component structure of an abstract quantum automaton:

```
specification =
      'automaton', WS, identifier, EOL,
      'control:', EOL, control specification,
      'actions (memory levels number = ', levels number,
            '):', EOL, action specification,
      'end';
```

```
levels number = ? number of quantum memory levels ?;
```

The meta-identifier `identifier` denotes unique names of specifying objects and it is used for naming automata in the previous rule. It is defined by the following way:

```
letter = ? all upper-case and lower-case Latin letters ?;
digit = ? all decimal digits ?;
identifier = letter, {letter | digit};
```

The meta-identifiers `WS` and `EOL` are described two kinds of delimiters. They are determined by the following rules:

```
WS = ? all white space characters ?;
EOL = ? control sequence "new line" ?;
```

The next rules are used to define the meta-identifier `control specification`.

```
control specification =
      entry node specification,
      node specification, {node specification};
entry node specification = 'entry', WS, node specification;
```

The meta-identifier `node specification` is the main linguistic unit to specify the control system of an automaton. Each node is described by its identifier and by its outgoing transition list.

```
node specification =
      identifier, '(', transition list, ')', EOL;
transition list =
      transition specification,
      {',', WS, transition specification};
```

Each transition τ is described by its label $\lambda(\tau)$ and its sink node $\mathrm{codom}(\tau)$.

```
transition specification = label, ':', WS, identifier;
```

Any identifier or the special symbol "`#`" can be used as a label. Note that the special symbol is used as the label "default" for nodes, which have only one outgoing transition.

```
label = '#' | identifier;
```

Now let's consider the last part of an automaton description, which is determined by the meta-identifier `action specification`.

Note that to determine an operator on a Hilbert space $\mathcal{H}_m$ it is sufficient to specify its action on vectors from an ortho-normal basis, for example, $|0\rangle, \ldots, |m-1\rangle$. So, we should specify vectors $|\Omega_0\rangle, \ldots, |\Omega_{m-1}\rangle$ from $\mathcal{H}_m \otimes l^2(\Lambda)$. The number of memory levels m determined by the meta-identifier `levels number` (see the rule for definition the meta-identifier `specification` above).

Thus, the meta-identifier `action specification` is defined by the next way:

```
action specification =
      identifier, ':', action, EOL,
      {identifier, ':', action, EOL};
```

In this rule the meta-identifier `identifier` refers on a node identifier.

```
action =
      '[', {index, ':', WS, 'S(', product-vector, ')', ','},
      index, ':', WS, 'S(', product-vector, ')', ']';
index = ? index of basis vector ?;
```

The meta-identifier `product-vector` is used for denoting the sum of elementary tensors, which corresponds to the image of the basis vector with index `index`.

```
product-vector =
  {index, ':', '|', function, '>', ','},
  index, ':', '|', function, '>';
function = {label, ':', complex, ','}, label, ':' complex;
complex = real | 'I', '*', real | real '+', 'I', '*', real;
real = ? real number ?;
```

Example 1. As example let's describe an automaton, which set a qubit in the state $|0\rangle$. Mathematical model for this automaton was described in [1, 2].

```
automaton QubitCleaner
control:
   entry measure(V0: exit, V1: flip)
   flip(#: exit)
actions (memory levels number = 2):
   measure: [0: S(0: |V0: 1>), 1: S(1: |V1: 1>)]
   flip: [0: S(1: |#: 1>), 1: S(0: |#: 1>)]
end
```

5 Trial Implementation of the Wireframe Model

Trial implementation of the model described above was performed by the authors and a group of Master students at the Department of Theoretical and Applied Computer Science at the V.N. Karazin Kharkiv National University.

As an implementation language was chosen language Java. This choice was due to the presence of a convenient free tool for rapidly develop compilers for Domain Specific Languages (ANTLR, see [8]) and a wide variety of free libraries for matrix calculations.

Table 1 shows libraries for matrix calculations, which were analysed in the process of working on the trial implementation.

Michael Thomas Flanagan Java Scientific Library was chosen for the trial implementation. This decision is grounded by the following constraints:

Table 1. Libraries for matrix calculations

Library Name	Library Location
COLT	http://acs.lbl.gov/software/colt/
Efficient Java Matrix Library (EJML)	http://code.google.com/efficient-java-matrix-library/
Java Matrix Library	http://jmatrices.sourceforge.net/index.html
Java Matrix Package (JAMA)	http://math.nist.gov/javanumerics/jama/
Michael Thomas Flanagan's Java Scientific Library	http://www.ee.ucl.ac.uk/~mflanaga/java/index.html
Universal Java Matrix Package	http://www.ujmp.org/

- the library should cover all matrix operations used under modelling abstract quantum automata;
- the library should provide interoperability with the library MPJ Express, which is an implementation of an MPI-like API used to write parallel Java applications for executing on a variety of parallel platforms ranging from multi-core processors to computing clusters/clouds [9].

The trial implementation of the wireframe model has revealed several problems, that need to be addressed:

1. processing time required to execute every step of quantum automaton has the exponential growth. Using parallel processing and grid computing can be considered as possible future solutions;
2. limited precision of computer processor may produce errors in the model of intermediate quantum memory states. Accumulation of these errors can be destroy correctness of physical postulates mapping. Application of symbolic computations can be considered as a possible future solution.

6 Conclusion

Summarising the above we can conclude:

- simulation wireframe model for studying quantum information processing systems is presented in the paper. This model is based on the notion of an abstract quantum automaton;
- the trial implementation of the wireframe model has performed. The realization detects a series of problems, which are described above;

– description language for quantum automaton "AQuanAut" has been developed.

Our nearest objective is re-engineering of the model to implement it basing on computing environment for high-performance computing and grid systems.

References

1. Alobaidi, M., Batyiv, A., Zholtkevych, G.: Abstract Quantum Automata as Formal Models of Quantum Information Processing Systems. In: V. Ermolayev et al.(eds.) ICT in Education, Research, and Industrial Applications. CCIS, vol. 347, pp. 19 – 38. Springer-Verlag, Berlin Heidelberg (2013)
2. Alobaidi, M., Batyiv, A., Zholtkevych, G.: Towards the Notion of an Abstract Quantum Automaton. arXiv:1204.3986v1 [cs.CC], http://arxiv.org/abs/1204.3986
3. Information technology – Object Management Group – Object Constraint Language (OCL). ISO/IEC 19507:2012(E)
4. Information technology – Syntactic metalanguage – Extended BNF. ISO/IEC 14977:1996(E)
5. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th Anniversary Edition. Cambridge University Press, Cambridge (2010)
6. OMG Unified Modeling Language^{TM}(OMG UML), Infrastructure. OMG (2011), http://www.omg.org/spec/UML/2.4.1/Infrastructure
7. OMG Unified Modeling Language^{TM}(OMG UML), Superstructure. OMG (2011), http://www.omg.org/spec/UML/2.4.1/Superstructure
8. Parr, T.: The Definitive ANTLR Reference. Building Domain-Specific Languages. Pragmatic Bookshelf, Raleigh, NC Dallas, TX (2007)
9. Shafi, A., Carpenter, B., Baker, M.: Nested parallelism for multi-core HPC systems using Java. J. Par. Distr. Comp., vol. 69, 6, 532–545 (2009)