Computer Communication & Collaboration (Vol. 2, Issue 2, 2014)
ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-002-24

Generators of Petri Net Models [1]

Dmitry A. Zaitsev

Computer Engineering Department of International Humanitarian University

Fontanskaya Doroga st., 33, Odessa 65009, Ukraine

E-mail: daze@acm.org

Abstract: To fill a gap in large-scale realistic Petri net models, meant for formal methods design and corresponding software testing, generators of Petri nets were presented. A Petri net generator is a program, developed on the parametric description of a Petri net, which produces a model having required characteristics, for instance, size. The technique of generators development was described on an example of computing grid model represented either in logical or in graphical format. Generators were successfully applied for testing plug-ins for simulating system Tina, verification of Ethernet protocols, and analysis of computing grids. Modeling of manufacturing systems, business processes and concurrent programs are prospective application areas of the generators.

Keywords: Infinite Petri net, Parametric description, Simulating system, Grid computing, Hypercube

1. Introduction

Petri nets find wide application in automated manufacture [1,2], business [3], telecommunication [4] and networking [5], computing [6]. But the majority of studied in literature Petri net examples [1-7] are rather simplified models of real-life systems and processes. A certain gap is formed between large-scale nets, employed in real-life projects, and illustrative examples, roaming among articles and monographs. Moreover, various simulating systems [7], which are counted in dozens and hundreds, are tested and presented on simplified examples of nets. Thus, there is a definite deficiency of the both realistic models and simulating systems which are able to analyze large-scale nets in admissible time. In many cases, detailed models, close to enterprise-class specifications, are a trade secret and simulating systems, which are able to analyze them, are spread on commercial ground.

So, certain difficulties arise, when developing formal methods of Petri nets analysis and the corresponding software, which are induced by the lack of actual nets. In many cases, the application of random Petri nets does not give the appreciated result, since artificial systems posses a series of peculiarities, for instance, are rather good decomposable into functional subnets (clans) [5] while random nets are close to indecomposable.

A demand is formed for a library of large-scale Petri nets, which are either models or specifications of real-life systems, as well as for facilities of automated constructing (synthesis) of specific Petri nets with given characteristics: number of vertices, density and localization of arcs, connectivity and so on. Owing to the lack of appropriate actual nets, it is difficult to formulate a set of characteristics and their ranges for various standard models: a parallel program; a computing,

[1] Described models and programs are put on web-site http://daze.ho.ua.

production, transport system etc. While there is no objective precondition for solving general tasks of Petri nets synthesis, it is possible to construct special generators of Petri nets for separate application domains.

The paper represents, in essence, a case study of a Petri net generator construction for modeling computing grids [8] but the described technique could be employed in wide range of Petri net application domains, including automated manufacture, business processes and programming.

2. Historical Notes and Formats of Files

During development of plug-ins for known simulating system Tina [9,10], named Deborah and Adriana [5] and destined for Petri nets decomposition into clans and compositional calculation of invariants correspondingly, software generators of Petri nets (fig. 1) were applied. They produced large-scale Petri nets which were thereafter used as tests. These nets had the structure of a simple chain, simple loop, and a sequence of connected basic fragments. Thus, the possibility of testing program modules on nets with dozens of thousands vertices was provided.

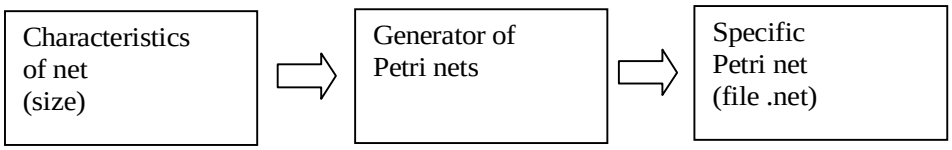

Figure 1 General scheme of a Petri nets generator operation

Analogous generators were used for verification of Ethernet protocols [5] and analysis of computing grids [11,12] with the aim of creating inductive base for generalizing obtained results on structures of an arbitrary size [13].

Remind that, Petri net is bipartite directed graph with a dynamic process defined on it [1]. One part of vertices is named places and drawn as circles, another part of vertices is named transitions and drawn as bars (rectangles), arcs could be multiple and their multiplicity is represented by integer numbers. Dynamic elements are named tokens and drawn as dots; they are situated inside places and move within net.

Simulating system Tina [9,10] accepts two formats of files describing a Petri net: logical (.net) and graphical (.ndr). For constructing generators either of formats could be used. Since for big Petri nets their visual representation becomes inessential, in the present paper, logical format is employed substantially. The further check visualization of a Petri net is done by embedded facilities of Tina that provide automatic constructing files with graphical format (drawing net).

A file with logical format (.net) contains an abstract description of a Petri net without information regarding its visual representation. Petri net graph is described as a list of transitions and their input and output arcs; initial marking is described as a list of places with values of their marking. Besides, Tina employs other information such as auxiliary labels of vertices, types of arcs, transitions' priorities and times of transitions' firing. This information is not used in the present work as far as a classical Petri net is generated, namely its graph required for structural analysis via calculation of places' and transitions' invariants. In the simplified form, the file format is described as follows

```
tr <t-name> <p-name>['*'<weight>],… -> <p-name>['*'<weight>],…
…
pl <p-name> (<marking>)
…
```

ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-002-24

```
net <net-name>
```

Transition description begins with reserved word "tr"; then the list of transition's incoming arcs follows, and after a delimiter "->" the list of outgoing arcs follows. An arc is described by a place name and arc's multiplicity (weight) indicated after a delimiter "*", multiplicity equal to unit is omitted. Compound names of vertices are parenthesized in curly brackets. An example of file is shown in Listing 2.

3. Parametric Model of a Square Grid

Let us study the technique of Petri nets generators construction on the example of an open square computing grid model [11,12]. Grid computing [8] represents a distributed computing taken to the next evolutionary level with the goal to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources aimed for solving large-scale practical tasks, weather forecast, for instance.

Grid node is a generalized device that implements processing and forwarding of packets. It contains a few ports; each port work in full duplex mode providing a separate independent channel for receiving and transmitting packets. Packets forwarding is based on the store-and-forward principle: arrived packed is stored into internal buffer and then forwarded to the destination port. The parametric description of a grid node $R^{i,j}$ model has the form (1); its graphical representation is shown in fig. 2.

$$\left(\left(\begin{pmatrix} to_u : pb_u, pol_u \rightarrow po_u, pbl \end{pmatrix} \\ \begin{pmatrix} ti_{u,v} : pi_u, pbl \rightarrow pb_v, pil_u \end{pmatrix}, \; v = \overline{1,4}, \; v \neq u \right), u = \overline{1,4}\right) \qquad (1)$$

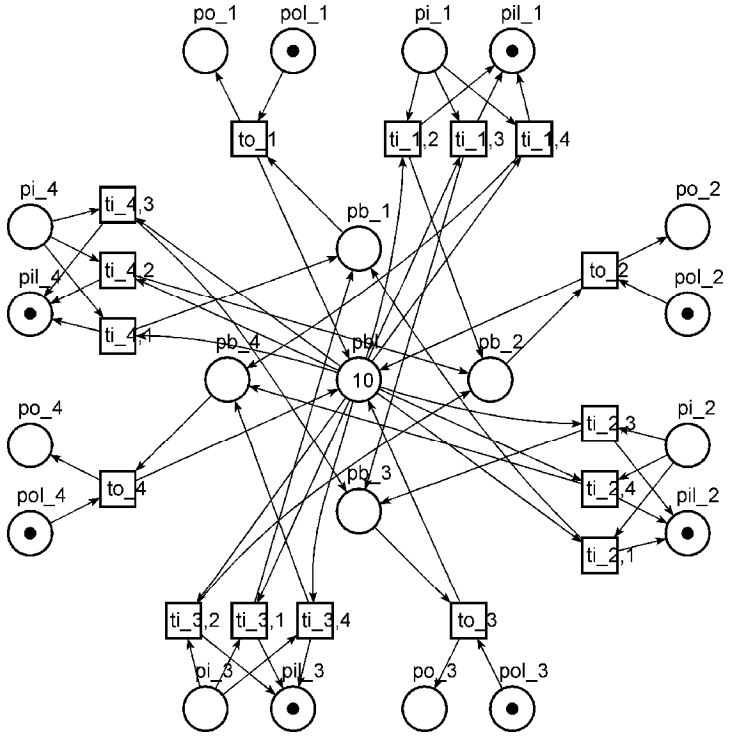

Figure 2 Grid node model

The transitions are the following: to_u – move a packet from the internal buffer to the port u buffer, $ti_{u,v}$ – forward a packet from port u input buffer to the internal buffer partition of port v. The places are the following: pi_u – buffer of the port u input channel, pil_u – buffer size of the port u input channel, po_u – buffer of the port u output channel, pol_u – buffer size of the port u output channel, pb_u – internal buffer partition for storing packets forwarded to the port u, pbl – the internal buffer size. Thus, the internal buffer is represented by five places: $pb_u, u = \overline{1,4}$ and pbl. The packets forwarding to the arrival port is not allowed. Each move of a packet within device model is accompanied with increment/decrement of available buffer size which is measured in number of packets. Contact places pi_u, pil_u, po_u, pol_u provide the composition of a grid.

To compose a square grid, nodes are situated within a square matrix and enumerated using the upper pair of indices (i, j) which specify the location of a device $R^{i,j}$ within grid. The composition of a grid is done via merging (fusing) contact places situated on the square sides of neighbor devices (fig. 3). Ports' enumeration in fig. 2 provides merging an input channel of a port with corresponding output channel of a port of a neighbor device and vice versa. The resulting Petri net is represented in parametric form (2); parameter k specifies the size of the grid.

ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-002-24

$$\left(\begin{array}{l}
\left(to_1^{i,j} : pol_1^{i,j}, pb_1^{i,j} -> po_1^{i,j}, pbl^{i,j}\right), \\
\left(ti_{1,v}^{i,j} : pi_1^{i,j}, pbl^{i,j} -> pil_1^{i,j}, pb_v^{i,j}\right), v = 2,3,4, \\
\left(to_3^{i,j} : pil_1^{i+1,j}, pb_3^{i,j} -> pi_1^{i+1,j}, pbl^{i,j}\right), \\
\left(ti_{3,v}^{i,j} : po_1^{i+1,j}, pbl^{i,j} -> pol_1^{i+1,j}, pb_v^{i,j}\right), v = 1,2,4, \\
\left(to_4^{i,j} : pol_4^{i,j}, pb_4^{i,j} -> po_4^{i,j}, pbl^{i,j}\right), \\
\left(ti_{4,v}^{i,j} : pi_4^{i,j}, pbl^{i,j} -> pil_4^{i,j}, pb_v^{i,j}\right), v = 1,2,3, \\
\left(to_2^{i,j} : pil_4^{i,j+1}, xb_2^{i,j} -> pi_4^{i,j+1}, pbl^{i,j}\right), \\
\left(ti_{2,v}^{i,j} : po_4^{i,j+1}, pbl^{i,j} -> pol_4^{i,j+1}, pb_v^{i,j}\right), v = 1,3,4
\end{array}\right), i = \overline{1,k}, j = \overline{1,k} \qquad (2)$$

The composition of the grid uses names of left and upper ports only (4 and 1) that is why contact places with port numbers 2 and 3 do not appear in (2). For instance, transitions of the third port are described as $to_3^{i,j} : pil_1^{i+1,j}, pb_3^{i,j} -> pi_1^{i+1,j}, pbl^{i,j}$ using the port names of the bottom neighboring device instead of the description $to_3^{i,j} : pol_1^{i,j}, pb_3^{i,j} -> po_1^{i,j}, pbl^{i,j}$ according to (1).

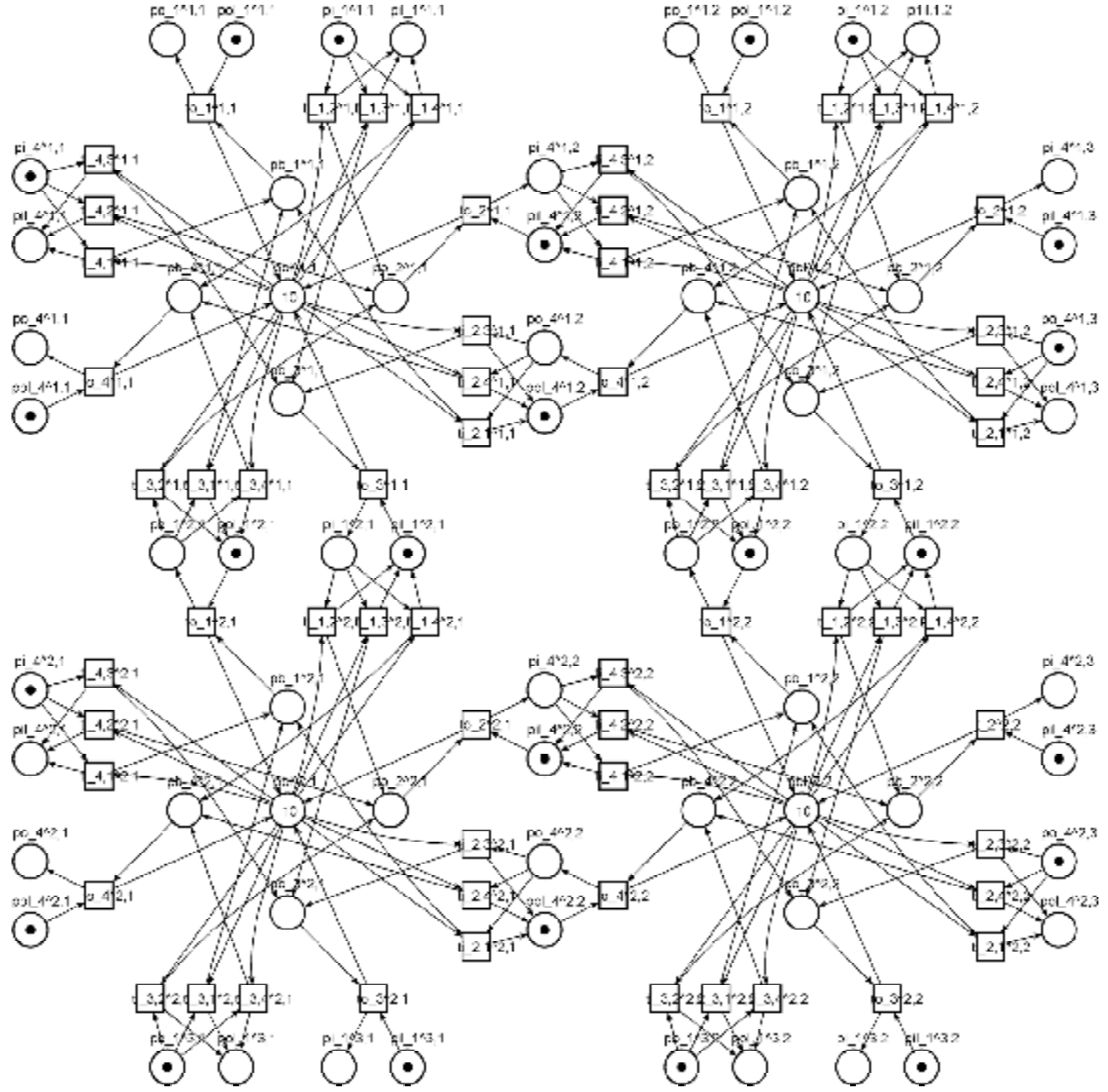

Figure 3 Composition of the open square grid model of size 2

4. Software Implementation of a Generator

The parametric description of grids via the form (2) is a source data for software implementation of Petri net generators. The generator of the open square grid models in programming language C is shown in Listing 1. The value of parameter k of the square grid size is inputted from the command line as argv[1].

Then, on each of used in (2) variables i, j, the corresponding loop is organized; because of small range of variable v, as an alternative to additional loops, the direct enumeration of all variants is employed. For printing by the program, the notation of net elements' names is modified avoiding subscript and superscript symbols; grid node index is written as a suffix in TEX-like notation: the lower index – after underscore character "_" and the upper index – after hat character "^". For instance, $ti_{1,3}^{5,7}$ looks like {ti_1,3^5,7}.

ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-002-24

```
/* generate open square grid on plane */

#include <stdio.h>

main( int argc, char * argv[] )
{
 int k,i,j;

 if( argc < 2 )
 {
   printf("*** USAGE: g2o k\n");
   return 2;
 }
 else k = atoi( argv[1] );

 for(i=1; i<=k; i++)
   for(j=1; j<=k; j++)
   {
     printf("tr {to_1^%d,%d} {pol_1^%d,%d} {pb_1^%d,%d} ->
       {po_1^%d,%d} {pbl^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {ti_1,2^%d,%d} {pi_1^%d,%d} {pbl^%d,%d} ->
       {pil_1^%d,%d} {pb_2^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {ti_1,3^%d,%d} {pi_1^%d,%d} {pbl^%d,%d} ->
       {pil_1^%d,%d} {pb_3^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {ti_1,4^%d,%d} {pi_1^%d,%d} {pbl^%d,%d} ->
       {pil_1^%d,%d} {pb_4^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {to_4^%d,%d} {pol_4^%d,%d} {pb_4^%d,%d} ->
       {po_4^%d,%d} {pbl^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {ti_4,1^%d,%d} {pi_4^%d,%d} {pbl^%d,%d} ->
       {pil_4^%d,%d} {pb_1^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {ti_4,2^%d,%d} {pi_4^%d,%d} {pbl^%d,%d} ->
       {pil_4^%d,%d} {pb_2^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {ti_4,3^%d,%d} {pi_4^%d,%d} {pbl^%d,%d} ->
       {pil_4^%d,%d} {pb_3^%d,%d}\n", i,j, i,j, i,j, i,j, i,j );
     printf("tr {to_2^%d,%d} {pil_4^%d,%d} {pb_2^%d,%d} ->
       {pi_4^%d,%d} {pbl^%d,%d}\n", i,j, i,j+1, i,j, i,j+1, i,j );
     printf("tr {ti_2,1^%d,%d} {po_4^%d,%d} {pbl^%d,%d} ->
       {pol_4^%d,%d} {pb_1^%d,%d}\n", i,j, i,j+1, i,j, i,j+1, i,j );
     printf("tr {ti_2,3^%d.%d} {po_4^%d,%d} {pbl^%d,%d} ->
       {pol_4^%d,%d} {pb_3^%d,%d}\n", i,j, i,j+1, i,j, i,j+1, i,j );
     printf("tr {ti_2,4^%d,%d} {po_4^%d,%d} {pbl^%d,%d} ->
       {pol_4^%d,%d} {pb_4^%d,%d}\n", i,j, i,j+1, i,j, i,j+1, i,j );
     printf("tr {to_3^%d,%d} {pil_1^%d,%d} {pb_3^%d,%d} ->
       {pi_1^%d,%d} {pbl^%d,%d}\n", i,j, i+1,j, i,j, i+1,j, i,j );
     printf("tr {ti_3,1^%d,%d} {po_1^%d^%d} {pbl^%d,%d} ->
       {pol_1^%d,%d} {pb_1^%d,%d}\n", i,j, i+1,j, i,j, i+1,j, i,j );
     printf("tr {ti_3,2^%d,%d} {po_1^%d,%d} {pbl^%d,%d} ->
       {pol_1^%d,%d} {pb_2^%d,%d}\n", i,j, i+1,j, i,j, i+1,j, i,j );
     printf("tr {ti_3,4^%d,%d} {po_1^%d,%d} {pbl^%d,%d} ->
       {pol_1^%d,%d} {pb_4^%d,%d}\n", i,j, i+1,j, i,j, i+1,j, i,j );
   }
 printf("net n2o%d\n", k);
}
```

Listing 1 Generator of the open square grid model

In Listing 2, Petri net generated by the program, shown in Listing 1, at the parameter value

$k = 2$, is represented; the net completely corresponds to the graphical representation of the grid model shown in fig. 3.

```
tr {to_1^1,1} {pol_1^1,1} {pb_1^1,1} -> {po_1^1,1} {pbl^1,1}
tr {ti_1,2^1,1} {pi_1^1,1} {pbl^1,1} -> {pil_1^1,1} {pb_2^1,1}
tr {ti_1,3^1,1} {pi_1^1,1} {pbl^1,1} -> {pil_1^1,1} {pb_3^1,1}
tr {ti_1,4^1,1} {pi_1^1,1} {pbl^1,1} -> {pil_1^1,1} {pb_4^1,1}
tr {to_4^1,1} {pol_4^1,1} {pb_4^1,1} -> {po_4^1,1} {pbl^1,1}
tr {ti_4,1^1,1} {pi_4^1,1} {pbl^1,1} -> {pil_4^1,1} {pb_1^1,1}
tr {ti_4,2^1,1} {pi_4^1,1} {pbl^1,1} -> {pil_4^1,1} {pb_2^1,1}
tr {ti_4,3^1,1} {pi_4^1,1} {pbl^1,1} -> {pil_4^1,1} {pb_3^1,1}
tr {to_2^1,1} {pil_4^1,2} {pb_2^1,1} -> {pi_4^1,2} {pbl^1,1}
tr {ti_2,1^1,1} {po_4^1,2} {pbl^1,1} -> {pol_4^1,2} {pb_1^1,1}
tr {ti_2,3^1.1} {po_4^1,2} {pbl^1,1} -> {pol_4^1,2} {pb_3^1,1}
tr {ti_2,4^1,1} {po_4^1,2} {pbl^1,1} -> {pol_4^1,2} {pb_4^1,1}
tr {to_3^1,1} {pil_1^2,1} {pb_3^1,1} -> {pi_1^2,1} {pbl^1,1}
tr {ti_3,1^1,1} {po_1^2^1} {pbl^1,1} -> {pol_1^2,1} {pb_1^1,1}
tr {ti_3,2^1,1} {po_1^2,1} {pbl^1,1} -> {pol_1^2,1} {pb_2^1,1}
tr {ti_3,4^1,1} {po_1^2,1} {pbl^1,1} -> {pol_1^2,1} {pb_4^1,1}
tr {to_1^1,2} {pol_1^1,2} {pb_1^1,2} -> {po_1^1,2} {pbl^1,2}
tr {ti_1,2^1,2} {pi_1^1,2} {pbl^1,2} -> {pil_1^1,2} {pb_2^1,2}
tr {ti_1,3^1,2} {pi_1^1,2} {pbl^1,2} -> {pil_1^1,2} {pb_3^1,2}
tr {ti_1,4^1,2} {pi_1^1,2} {pbl^1,2} -> {pil_1^1,2} {pb_4^1,2}
tr {to_4^1,2} {pol_4^1,2} {pb_4^1,2} -> {po_4^1,2} {pbl^1,2}
tr {ti_4,1^1,2} {pi_4^1,2} {pbl^1,2} -> {pil_4^1,2} {pb_1^1,2}
tr {ti_4,2^1,2} {pi_4^1,2} {pbl^1,2} -> {pil_4^1,2} {pb_2^1,2}
tr {ti_4,3^1,2} {pi_4^1,2} {pbl^1,2} -> {pil_4^1,2} {pb_3^1,2}
tr {to_2^1,2} {pil_4^1,3} {pb_2^1,2} -> {pi_4^1,3} {pbl^1,2}
tr {ti_2,1^1,2} {po_4^1,3} {pbl^1,2} -> {pol_4^1,3} {pb_1^1,2}
tr {ti_2,3^1.2} {po_4^1,3} {pbl^1,2} -> {pol_4^1,3} {pb_3^1,2}
tr {ti_2,4^1,2} {po_4^1,3} {pbl^1,2} -> {pol_4^1,3} {pb_4^1,2}
tr {to_3^1,2} {pil_1^2,2} {pb_3^1,2} -> {pi_1^2,2} {pbl^1,2}
tr {ti_3,1^1,2} {po_1^2^2} {pbl^1,2} -> {pol_1^2,2} {pb_1^1,2}
tr {ti_3,2^1,2} {po_1^2,2} {pbl^1,2} -> {pol_1^2,2} {pb_2^1,2}
tr {ti_3,4^1,2} {po_1^2,2} {pbl^1,2} -> {pol_1^2,2} {pb_4^1,2}
tr {to_1^2,1} {pol_1^2,1} {pb_1^2,1} -> {po_1^2,1} {pbl^2,1}
tr {ti_1,2^2,1} {pi_1^2,1} {pbl^2,1} -> {pil_1^2,1} {pb_2^2,1}
tr {ti_1,3^2,1} {pi_1^2,1} {pbl^2,1} -> {pil_1^2,1} {pb_3^2,1}
tr {ti_1,4^2,1} {pi_1^2,1} {pbl^2,1} -> {pil_1^2,1} {pb_4^2,1}
tr {to_4^2,1} {pol_4^2,1} {pb_4^2,1} -> {po_4^2,1} {pbl^2,1}
tr {ti_4,1^2,1} {pi_4^2,1} {pbl^2,1} -> {pil_4^2,1} {pb_1^2,1}
tr {ti_4,2^2,1} {pi_4^2,1} {pbl^2,1} -> {pil_4^2,1} {pb_2^2,1}
tr {ti_4,3^2,1} {pi_4^2,1} {pbl^2,1} -> {pil_4^2,1} {pb_3^2,1}
tr {to_2^2,1} {pil_4^2,2} {pb_2^2,1} -> {pi_4^2,2} {pbl^2,1}
tr {ti_2,1^2,1} {po_4^2,2} {pbl^2,1} -> {pol_4^2,2} {pb_1^2,1}
tr {ti_2,3^2.1} {po_4^2,2} {pbl^2,1} -> {pol_4^2,2} {pb_3^2,1}
tr {ti_2,4^2,1} {po_4^2,2} {pbl^2,1} -> {pol_4^2,2} {pb_4^2,1}
tr {to_3^2,1} {pil_1^3,1} {pb_3^2,1} -> {pi_1^3,1} {pbl^2,1}
tr {ti_3,1^2,1} {po_1^3^1} {pbl^2,1} -> {pol_1^3,1} {pb_1^2,1}
tr {ti_3,2^2,1} {po_1^3,1} {pbl^2,1} -> {pol_1^3,1} {pb_2^2,1}
tr {ti_3,4^2,1} {po_1^3,1} {pbl^2,1} -> {pol_1^3,1} {pb_4^2,1}
tr {to_1^2,2} {pol_1^2,2} {pb_1^2,2} -> {po_1^2,2} {pbl^2,2}
tr {ti_1,2^2,2} {pi_1^2,2} {pbl^2,2} -> {pil_1^2,2} {pb_2^2,2}
tr {ti_1,3^2,2} {pi_1^2,2} {pbl^2,2} -> {pil_1^2,2} {pb_3^2,2}
tr {ti_1,4^2,2} {pi_1^2,2} {pbl^2,2} -> {pil_1^2,2} {pb_4^2,2}
tr {to_4^2,2} {pol_4^2,2} {pb_4^2,2} -> {po_4^2,2} {pbl^2,2}
tr {ti_4,1^2,2} {pi_4^2,2} {pbl^2,2} -> {pil_4^2,2} {pb_1^2,2}
tr {ti_4,2^2,2} {pi_4^2,2} {pbl^2,2} -> {pil_4^2,2} {pb_2^2,2}
tr {ti_4,3^2,2} {pi_4^2,2} {pbl^2,2} -> {pil_4^2,2} {pb_3^2,2}
tr {to_2^2,2} {pil_4^2,3} {pb_2^2,2} -> {pi_4^2,3} {pbl^2,2}
tr {ti_2,1^2,2} {po_4^2,3} {pbl^2,2} -> {pol_4^2,3} {pb_1^2,2}
tr {ti_2,3^2.2} {po_4^2,3} {pbl^2,2} -> {pol_4^2,3} {pb_3^2,2}
tr {ti_2,4^2,2} {po_4^2,3} {pbl^2,2} -> {pol_4^2,3} {pb_4^2,2}
tr {to_3^2,2} {pil_1^3,2} {pb_3^2,2} -> {pi_1^3,2} {pbl^2,2}
tr {ti_3,1^2,2} {po_1^3^2} {pbl^2,2} -> {pol_1^3,2} {pb_1^2,2}
tr {ti_3,2^2,2} {po_1^3,2} {pbl^2,2} -> {pol_1^3,2} {pb_2^2,2}
```

Computer Communication & Collaboration (Vol. 2, Issue 2, 2014)
ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-002-24

```
tr {ti_3,4^2,2} {po_1^3,2} {pbl^2,2} -> {pol_1^3,2} {pb_4^2,2}
net n2o2
```

Listing 2 Generated model of the open square grid of size 2 (.net format)

Thus, the technique of constructing a Petri net generator in logical format is specified as follows:

(a) obtain a parametric description of infinite Petri net;

(b) compose program loops according to the indices variation of the parametric description;

(c) compose printed lines according to the lines of the parametric description;

(d) check nets generated for definite values of parameters in system Tina.

5. Application of Generated Nets

The main application area of specific grid models, obtained as result of generators work, is the formation of data base of actual nets for further inductive conclusions regarding the properties of infinite Petri nets with regular structure.

On basis of calculation and analysis of place invariants for a sequence of grid models with definite sizes [11,12] the general parametric description of invariants was obtained and the p-invariance of Petri net for an arbitrary value of parameter k was proven. Analogous results were obtained in [13] for hypercube structures of an arbitrary size with an arbitrary number of dimensions. For instance, the parametric description of place invariants of the grid (2) has the form (3).

Remind, that place invariants [5] are nonnegative integer weights of places such that the weighted sum of tokens remains constant in every reachable marking i.e. invariant net is structurally safe (conservative) and bounded. Invariants are solutions of linear homogeneous system of equations constructed as balance of input and output arcs for each transition [11]. For instance, on parametric row $to_1^{i,j} : pol_1^{i,j}, pb_1^{i,j} -> po_1^{i,j}, pbl^{i,j}$ of (2), the equations $-xpol_1^{i,j} - xpb_1^{i,j} + xpo_1^{i,j} + xpbl^{i,j} = 0$ were constructed.

$$\left(\begin{array}{l}
(pi_1^{i,j}, pil_1^{i,j}), i = \overline{1,k+1},\ j = \overline{1,k};\\
(po_1^{i,j}, pol_1^{i,j}), i = \overline{1,k+1},\ j = \overline{1,k};\\
(pi_4^{i,j}, pil_4^{i,j}), i = \overline{1,k},\ j = \overline{1,k+1};\\
(po_4^{i,j}, pol_4^{i,j}), i = \overline{1,k},\ j = \overline{1,k+1};\\
(pb_1^{i,j}, pb_2^{i,j}, pb_3^{i,j}, pb_4^{i,j}, pbl^{i,j}), i = \overline{1,k},\ j = \overline{1,k};\\
(((pil_1^{i,j}, pol_1^{i,j}, pil_4^{i,j}, pol_4^{i,j}, pbl^{i,j}), i = \overline{1,k},\ j = \overline{1,k};),\\
\quad ((pil_4^{i,k+1}, pol_4^{i.k+1}), i = \overline{1,k}), ((pil_1^{k+1,j}, pol_1^{k+1.j}), j = \overline{1,k}))\\
(((pi_1^{i,j}, po_1^{i,j}, pi_4^{i,j}, po_4^{i,j}, ((pb_u^{i,j}), u = \overline{1,4};)), i = \overline{1,k},\ j = \overline{1,k};),\\
\quad ((pi_4^{i,k+1}, po_4^{i.k+1}), i = \overline{1,k}), ((pi_1^{k+1,j}, po_1^{k+1.j}),\ j = \overline{1,k}))
\end{array}\right) \quad (3)$$

Parametric representation (3) of basis invariants matrix is rather sophisticated because it is valid for any given magnitude of parameter k. Only nonzero elements are listed, and in the example of

square grid, all of them are equal to unit. There are two types of parametric rows: each of rows 1-5 describes a set of rows with a few nonzero elements (two for rows 1-4 and five for row 5); each of rows 6 and 7 describes a single row containing a set of nonzero elements. To distinguish difference, brackets are used.

Invariants calculated by system Tina for definite values of parameter k coincide with invariants generated from (3). Thus, the described technique could be used for generating net invariants on their parametric description. For instance, the place invariants of the net, shown in fig. 3, are represented in Listing 3 in an explicit form.

```
({pi_1^1,1},{pil_1^1,1})        ({po_1^1,1},{pol_1^1,1})
({pi_1^1,2},{pil_1^1,2})        ({po_1^1,2},{pol_1^1,2})
({pi_1^2,1},{pil_1^2,1})        ({po_1^2,1},{pol_1^2,1})
({pi_1^2,2},{pil_1^2,2})        ({po_1^2,2},{pol_1^2,2})
({pi_1^3,1},{pil_1^3,1})        ({po_1^3,1},{pol_1^3,1})
({pi_1^3,2},{pil_1^3,2})        ({po_1^3,2},{pol_1^3,2})
({pi_4^1,1},{pil_4^1,1})        ({po_4^1,1},{pol_4^1,1})
({pi_4^1,2},{pil_4^1,2})        ({po_4^1,2},{pol_4^1,2})
({pi_4^1,3},{pil_4^1,3})        ({po_4^1,3},{pol_4^1,3})
({pi_4^2,1},{pil_4^2,1})        ({po_4^2,1},{pol_4^2,1})
({pi_4^2,2},{pil_4^2,2})        ({po_4^2,2},{pol_4^2,2})
({pi_4^2,3},{pil_4^2,3})        ({po_4^2,3},{pol_4^2,3})
({pb_1^1,1},{pb_2^1,1},{pb_3^1,1},{pb_4^1,1},{pbl^1,1})
({pb_1^1,2},{pb_2^1,2},{pb_3^1,2},{pb_4^1,2},{pbl^1,2})
({pb_1^2,1},{pb_2^2,1},{pb_3^2,1},{pb_4^2,1},{pbl^2,1})
({pb_1^2,2},{pb_2^2,2},{pb_3^2,2},{pb_4^2,2},{pbl^2,2})
({pil_1^1,1},{pol_1^1,1},{pil_4^1,1},{pol_4^1,1},{pbl^1,1},
 {pil_1^1,2},{pol_1^1,2},{pil_4^1,2},{pol_4^1,2},{pbl^1,2},
 {pil_1^2,1},{pol_1^2,1},{pil_4^2,1},{pol_4^2,1},{pbl^2,1},
 {pil_1^2,2},{pol_1^2,2},{pil_4^2,2},{pol_4^2,2},{pbl^2,2},
 {pil_1^3,1},{pol_1^3,1},{pil_1^3,2},{pol_1^3,2},
 {pil_4^1,3},{pol_4^1,3},{pil_4^2,3},{pol_4^2,3})
({pi_1^1,1},{po_1^1,1},{pi_4^1,1},{po_4^1,1},
 {pb_1^1,1},{pb_2^1,1},{pb_3^1,1},{pb_4^1,1},
 {pi_1^1,2},{po_1^1,2},{pi_4^1,2},{po_4^1,2},
 {pb_1^1,2},{pb_2^1,2},{pb_3^1,2},{pb_4^1,2},
 {pi_1^2,1},{po_1^2,1},{pi_4^2,1},{po_4^2,1},
 {pb_1^2,1},{pb_2^2,1},{pb_3^2,1},{pb_4^2,1},
 {pi_1^2,2},{po_1^2,2},{pi_4^2,2},{po_4^2,2},
 {pb_1^2,2},{pb_2^2,2},{pb_3^2,2},{pb_4^2,2},
 {pi_1^3,1},{po_1^3,1},{pi_1^3,2},{po_1^3,2},
 {pi_4^1,3},{po_4^1,3},{pi_4^2,3},{po_4^2,3})
```

Listing 3 Place invariants of the open square grid with size 2 (fig. 2)

In some cases, the automatic visualization of generated nets is useful for evaluating general

ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-002-24

patterns of layout and additional check of composition rules. In fig. 4, model of grid of size 4 is represented in graphical form that is created automatically by simulating system Tina.

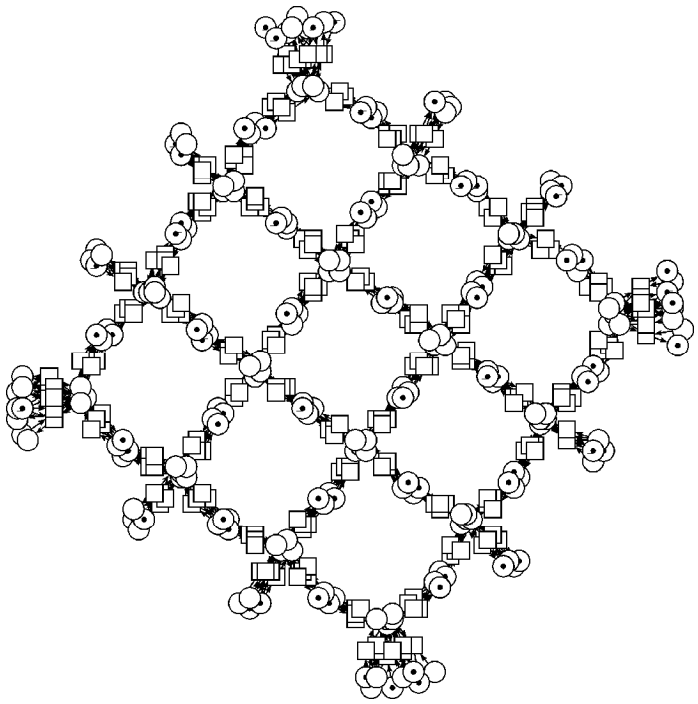

Figure 4 Automatic visualization of a generated grid model in Tina

Moreover, generated Petri net models could be applied as tests in Petri net simulating and analysis systems development, especially, when the properties of studied nets are known a priori. For instance, considered models of grids are safe according to the composition rules because each transition has exactly two input and two output places. It allows debugging software on big dimension nets.

6. Generating Petri Nets in Graphical Form

Automatic visualization of grid models is not distinguished by clearness and allows visual evaluation of the general pattern only. Some applications require working with graphical formats, for instance, for watching token game and studying transitions firing sequences. A model could be generated in graphical format as well.

Graphical file format (.ndr) of system Tina contains such extra information, regarding described in section 2 logical format (.net), as coordinates of vertices on a plane and arcs' curves description:

```
t <xpos> <ypos> <transition-name> <options>
p <xpos> <ypos> <place-name> <marking> <options>
e <vertex1-name> <vertex2-name> <weight> <options>
e <vertex1-name> <ang> <rad> <vertex2-name> <ang> <rad> <options>
```

Lines of types ‘p’ and ‘t’ describe places and transitions correspondently. Each row contains node coordinates (xpos, ypos) on a plane and its name; place description also includes its initial marking. Each arc is described separately with a line of type ‘e’. It contains names of the source and target vertices and multiplicity (weight) for a straight arc; for a curved arc, two pairs of additional parameters (ang, rad) define an angle and radius of arc curve for its both ends.

The technique of generating .ndr files starts with drawing in Tina a model of a grid node (fig. 1) and saving the corresponding .ndr file. An example of a net fragment description containing a place, a transition and an arc connecting them, for grid node model shown in fig. 1, looks like:

```
p 200.0 50.0 {po_1} 0 n
t 240.0 130.0 {to_1} 0 w n
e {to_1} {po_1} 1 n
```

Then the obtained .ndr file is used as a pattern to fill in the main loop of the generator which overall organization is similar to Listing 1. Only descriptions of vertices contain coordinates which should be recalculated for each node of the grid on its indices; relative format of arcs’ curvature description does not require their correction. Based on an element description, a format string of corresponding printf operator is created. Instead of definite coordinates (xpos, ypos) it contains format “%.1f %.1f” for printing recalculated coordinates and a format “^%d.%d” is inserted into vertices names for printing current node index within grid:

```
printf("p %.1f %.1f {po_1^%d.%d} 0 n\n",(i-1)*DI+200.0, (j-1)*DJ+50.0, j, i);
printf("t %.1f %.1f {to_1^%d.%d} 0 w n\n",(i-1)*DI+240.0, (j-1)*DJ+130.0, j, i);
printf("e {to_1^%d.%d} {po_1^%d.%d} 1 n\n",j, i, j, i);
```

Coordinates of vertices are calculated based on vertical and horizontal grid node offsets DI and DJ correspondingly which are equal to the maximal coordinates of a single grid node description. Then the local offset of a vertex within grid node is added. Reverse order of indices for coordinates regarding names is explained by the fact that in the grid model a matrix order of indexing is used: first index (i) – number of a row, second index (j) – number of a column. But in Tina, the first coordinate axis (x) is horizontal and the second (y) is vertical.

Series of grid models generators, working in graphical format, were developed and employed in investigation of infinite Petri nets properties [11-13]. An example of generated open square grid model of size 4 is shown in fig. 5.

Thus, the technique of constructing a Petri net generator in graphical format is specified as follows:

(a) obtain a parametric description of infinite Petri net;

(b) compose basic fragments of Petri net in system Tina in graphical form;

(c) write programs to generate basic fragments of Petri net in graphical format using saved net fragments;

(d) compose program loops according to the indices variation of the parametric description;

(e) insert printed lines of the basic fragments;

(f) add vertical and horizontal offsets depending on the parameters of loops to the coordinates of Petri net elements;

(g) check nets generated for definite values of parameters in system Tina.

ISSN 2292-1028(Print) 2292-1036(Online)
DOIC: 2292-1036-2014-02-002-24

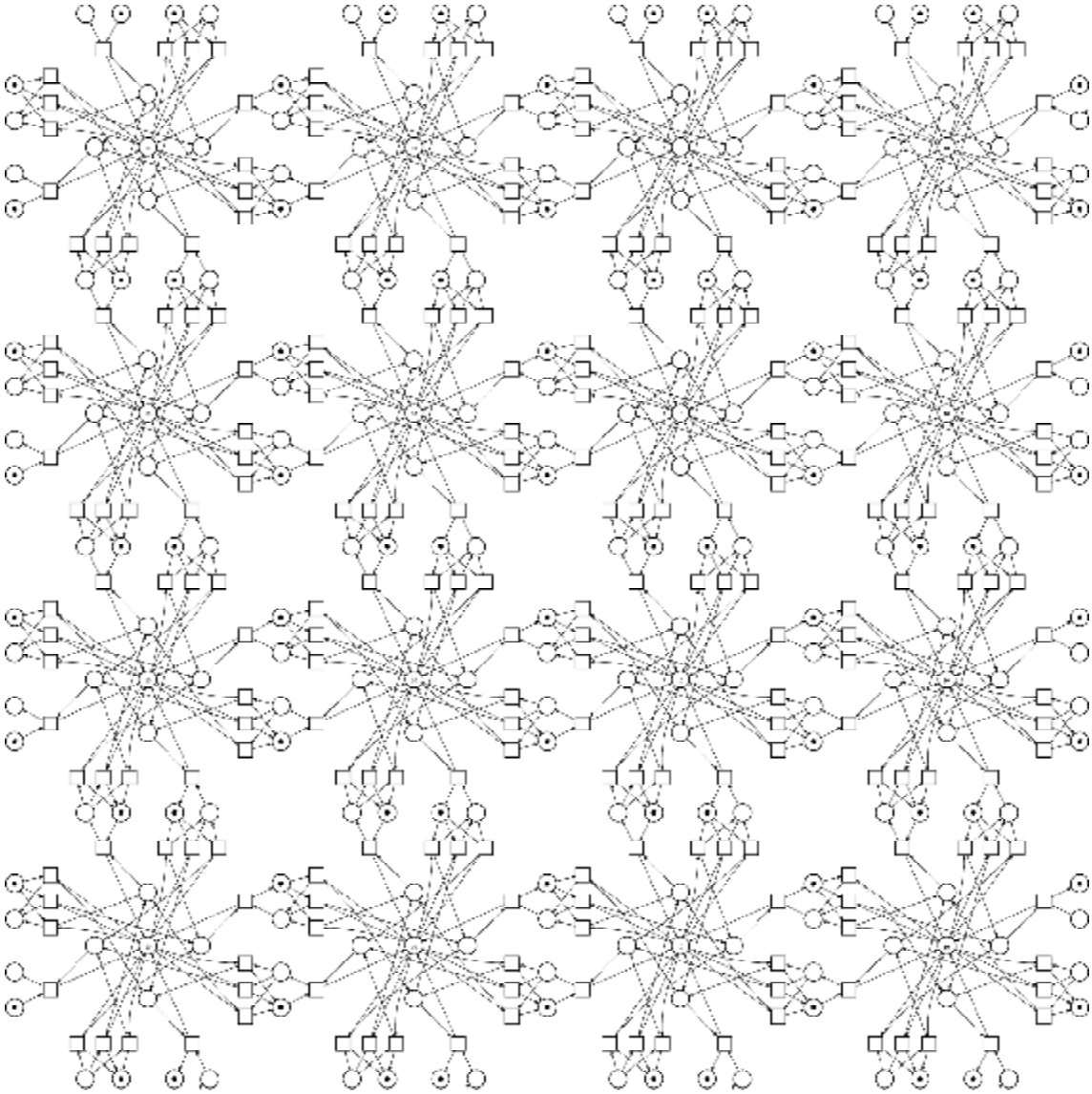

Figure 5 An example of the open square grid model of size 4 generated in graphical format

7. Conclusions

The technique of constructing software generators of Petri nets with regular structure on basis of their parametric description was presented.

The technique was applied successfully for testing modules Deborah and Adriana for compositional analysis [5] of Petri nets, verification of Ethernet protocols, and analysis of flat computing grids [11,12] and computing structures of hypercube [13]. The application of the technique allowed drawing conclusions regarding the properties of infinite Petri nets which necessity of use is demanded by the development of computer and communication technology considering interaction of unlimited number of devices.

The technique could be employed as well in wide range of Petri net application domains, including automated manufacture, business processes and programming.

References

[1]. Li, Z.W., and Al-Ahmari, A.M. (Eds.) (2013), *Formal Methods in Manufacturing Systems: Recent Advances*. IGI-Global.

[2]. Li, Z.W., and Zhou, M.C. (2010), *Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach*. New York: Springer-Verlag.

[3]. Aalst, W., and Stahl, C. (2011), *Modeling Business Processes: A Petri Net-Oriented Approach. Cambridge*, MA: MIT Press.

[4]. Girault, C. and Valk, R. (2003), *Petri Nets for Systems Engineering-A Guide to Modeling, Verification, and Applications*. Berlin, Germany: Springer-Verlag.

[5]. Zaitsev, D.A. (2013), *Clans of Petri Nets: Verification of protocols and performance evaluation of networks*. LAP LAMBERT Academic Publishing.

[6]. Best, E., Devillers, R., and Koutny, M. (2001), *Petri Net Algebra*. New York: Springer-Verlag.

[7]. Petri Nets World: http://www.informatik.uni-hamburg.de/TGI/PetriNets/

[8]. Information Resources Management Association (USA) (2012) *Grid and Cloud Computing: Concepts, Methodologies, Tools and Applications (4 Vol.)*. IGI-Global.

[9]. Berthomieu, B., Ribet, O.-P., and Vernadat, F. (2004), "The tool TINA-construction of abstract state space for Petri nets and time Petri nets". *International Journal of Production Research*, 42(4).

[10]. Petri Net Simulating System Tina: http://www.laas.fr/tina

[11]. Shmeleva, T.R., Zaitsev, D.A., and Zaitsev, I.D. (2009), "Analysis of Square Communication Grids via Infinite Petri Nets". *Transactions of Odessa National Academy of Telecommunication*, no. 1: 27-35.

[12]. Zaitsev, D.A. (2013), "Verification of Computing Grids with Special Edge Conditions by Infinite Petri Nets". *Automatic Control and Computer Sciences*, 47(7): 403–412.

[13]. Zaitsev, D.A., and Shmeleva, T.R. (2010), "Verification of hypercube communication structures via parametric Petri nets". *Cybernetics and Systems Analysis*, 46(1): 105-114.