Analysis of the Operational Petri Nets by a Distributed System

Andrei G. Karatkevich, Tomasz Gratkowski

Abstract - **In this paper a method of dynamic analysis of big operational Petri nets is described. A net is decomposed and its blocks are distributed within a computer network. Each block is simulated independently, and the results of simulation are joined and interpreted by the master computer.**

Keywords – **Operational Petri nets, distributed systems, Java, JPVM, simulation.**

I. Introduction

Petri nets [4] and Petri-net-based languages (such as Grafcet and SFC [3]) are a tool, both practical and theoretical, widely used for modeling and description of digital devices, parallel algorithms, communication protocols, industrial control systems, local-area networks and, generally, hardware and software concurrent systems [1,4]. So, methods of analysis of Petri nets have great practical importance. But as far as such concurrent systems may have huge number of reachable states (the so-called state explosion problem), analysis of big nets is a difficult task. In some cases it can be simplified by net decomposition [7]. A Petri net is a distributed system in course of its nature; so it seems to be convenient to analyze big Petri nets in a distributed way by a computer network. The method of such analysis is described in the paper.

II. Operational Petri Nets

A *Petri net* (PN) [4] can be represented as a bipartite oriented graph with two kinds of nodes called *places* and *transitions*. This is a dynamic model; a current state M of a net is called *marking* and is specified by the *tokens* contained in the places. Net marking can be changed by the transition *firing* according to the simple rules (a transition can fire if all its input places contain tokens; transition firing removes a token from every input place and adds a token to every output place).

The *operational Petri nets* are defined in [6,7]. In such nets he non-intersecting sets of *input* and *output* places (I and O, respectively) are selected; in the initial marking M_0 only the input places can have tokens. A correct operational PN satisfies the next conditions.

1. The net is *safe* (for every reachable marking no place contains more than one token).
2. From any marking M reachable from M_0, a *dead* marking is reachable (where no transition can fire).
3. All the reachable dead markings are *terminal* (only the output places contain tokens).

At Fig. 1 an example of operational PN is shown. It is correct for the initial markings {1, 2} and {3}.

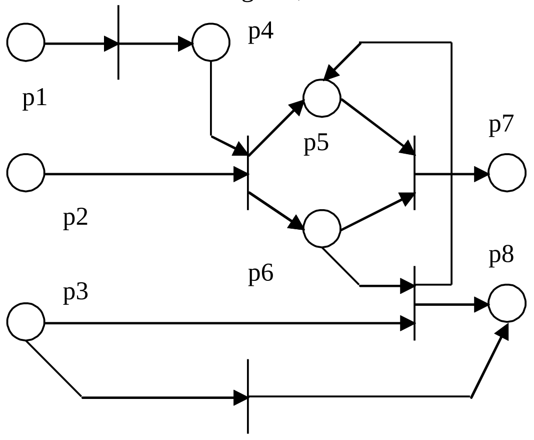

Fig.1 An example of operational Petri net

An operational PN can be used as a specification of a structure of parallel algorithm, which is initiated by some correct input states and terminates attaining one of the output states. In particular, such net may have only one input place and one output place. Such two-pole structures are widespread in the traditional approaches of organization of computational and control algorithms [7].

II. Analysis of Operational Petri Nets

There are two main tasks of analysis of the operational Petri nets: to check whether given net is correct for given initial marking and, if it is correct, to detect which terminal markings are reachable from the initial marking [6,7]. Both tasks can be solved by constructing the complete state space of the net, but the state space size depends exponentially of the size of net (the so-called state explosion problem), and there are practical examples of nets with hundreds of places and transitions. It is easy to see that such nets practically cannot be analyzed by investigation of complete state spaces.

One of the prospective approaches of overcoming such difficulties is net decomposition [7]. In [6] the method of block decomposition of an operational Petri net is proposed and analysis method based on it. Below it is described in short.

Two transitions are in the relation of *alternative joint* if their input or output sets of places intersect. Transitive closure of the relation of alternative joint of the set of transitions of an operational PN specifies its decomposition into minimum *blocks*. If blocks T_i and T_j have a common place which is output for T_i and input for T_j, then T_j cannot be analyzed before T_i. It specifies relation V between the blocks, which after uniting the cycles (item 2 of the algorithm below) becomes a relation of partial order. The analysis algorithm looks as follows [6].

Andrei Karatkevich, Tomasz Gratkowski - University of Zielona Góra, ul. Podgórna 50, 65-246 Zielona Góra, POLAND. E-mail: <A.Karatkevich,T.Gratkowski>@iie.uz.zgora.pl.
The work is partially supported by Polish State Committee for Scientific Research (KBN) grant № 4T11C 006 24

Algorithm 1

```
Input: an operational PN T and its initial marking $M_0$.
Output: set of terminal markings, if the net is correct.
1. Decompose the net T into minimum blocks.
2. Unite all the cycles composed by blocks.
3. D := {$M_0$}.
4. While D contains markings in which some tokens are situated in the non-output places of T, do:
   4.1. Find a block $T_i$ such that there is no block $T_j$ not analyzed yet for which $(T_j,T_i)\in V$.
   4.2. For each marking M belonging to D such that there are tokens in the input nodes of $T_i$, find the terminal markings of $T_i$ reachable from it and replace M in D by those markings (tokens not belonging to the places of $T_i$ do not change their positions).
   4.3. If at least one of the initial markings for $T_i$ is found to be incorrect, go to 6.
5. The initial marking $M_0$ is correct and D contains all the terminal markings reachable from it in T. The end.
6. The initial marking is incorrect. The end.
```

For proof of correctness of the algorithm and the experimental results see [6].

It is easy to see that block structure (obtained after performing steps 1 and 2 of Algorithm 1) specifies the partial order of analysis of the fragments of net. Hence, some of the blocks could be analyzed in parallel, if there is a possibility of multiple threads. In [2] the parallelized version of Algorithm 1 is described. It is intended for a multiprocessor structure with common memory. But for the big nets their analysis would be more efficient by using computer networks, for which the algorithms mentioned above cannot be applied directly. In this paper the modification intended for distributed analysis in the networks is described.

III. Distributed Analysis

The problem with distributed version of the algorithm is that although analysis of the blocks can be performed in parallel by different processors, at every step of analysis a set of markings of the whole net (set D in Algorithm 1) should be kept; in our method it is kept in a centralized way. So, we have chosen the next decision: there is a master computer which keeps description of set D and controls the analysis process. The rest of computers in the network are slaves; each slave has in its memory description of some subnets (blocks) of the Petri net and, obtaining from the master the sets of possible initial markings of the subnets, performs their analysis. The master obtains from the slaves the sets of possible terminal markings of their subnets (and eventually the messages about incorrectness) and controls the analysis process.

Before starting the net analysis, the blocks of the decomposed net should be assigned to the processors in a way which decreases the analysis time. According to the structure we have selected, there is no direct communication between slaves; the whole inter-computer communication goes through the master. We suppose that number of input and output places of the blocks is much less than their whole number of nets (that is a usual situation in practical applications), hence communication time is not taken into account here (but, according to the experimental results, it does affect the analysis tome noticeably, which should be taken into account in further work). For optimal partitioning the partial order in which blocks have to be analyzed and evaluated analysis time of each block should be considered.

We use the partitioning algorithm described in [5]. For evaluation of time-complexity of a block analysis the next empirical formula is used: $\mathrm{time}(T_i) \propto 3^{|P_i|/6}$, where $\mathrm{time}(T_i)$ is analysis time of block T_i, and P_i is the set of places of T_i. Using this evaluation, the program dependence graph (PDG, [5]) is built having the same structure as the directed acyclic graph specifying relation V, and to every node i the weight w_i is assigned according to the next formula:

$$w_i = 3^{|P_i|/6} \tag{1}$$

The method of distributed analysis is described below.

Algorithm 2a (initialization).

```
Input: an operational PN T.
Output: a processor assignment.
1. Decompose the net T into minimum blocks.
2. Unite all the cycles composed by blocks.
3. Create a PDG: nodes correspond to the blocks; arcs correspond to the relation V between the blocks. To every node a weight is assigned according to (1).
4. Build a parse tree for the PDG [5].
5. Perform processor assignment (by using the algorithm from [5]).
6. Write to every processor's memory descriptions of the blocks corresponding to the assigned nodes of PDG.
```

Algorithm 2b (master)

```
Input: an initial marking $M_0$; graph of block structure; processor assignment.
Output: set of terminal markings, if the net is correct; otherwise a message that it is incorrect.
1. D := {$M_0$}.
2. While not all the blocks are analyzed, do:
   2.1. Find a block $T_i$ such that there is no block $T_j$ not analyzed yet for which $(T_j,T_i)\in V$.
   2.2. For each marking M belonging to D such that there are tokens in the input nodes of $T_i$, send $M(T_i)$ to the processor to which the block is assigned.
   2.3. If from any slave the description of terminal markings of block $T_i$ reachable from $M(T_i)$ is obtained, replace in D the markings corresponding to $M(T_i)$ by the markings corresponding to the obtained terminal sub-markings.
   2.4. If from any slave the message is obtained that a block is incorrect, go to 4.
3. The initial marking $M_0$ is correct and D contains all the terminal markings reachable from it in T. The end
4. The initial marking is incorrect. The end.
```

Algorithm 2c (slave)

Input: assigned blocks of a net; initial markings of the blocks.

Output: set of terminal markings of the blocks, if the blocks are correct; otherwise a message that a block is incorrect.

When from the master description of an initial marking of block T_i ($M_0(T_i)$) is obtained:

1. Find all the terminal markings reachable from $M_0(T_i)$.
2. If T_i is correct for $M_0(T_i)$, send to master description of the terminal markings. Else send the message that T_i is incorrect.

IV. Implementation of the method

The described method has been implemented in Java. This programming language has been chosen due to its next points: availability of the mechanisms supporting distributed programming, support of multithreading and availability of parsers for XML language. As a distributed environment for the project Java Parallel Virtual Machine (JPVM) [8] was selected. Advantages of JPVM are its interface close to the interface of Parallel Virtual Machine (PVM) [9] and semantics fitting for the object-oriented programming language Java. JPVM has been completely implemented in Java, so it can be used in a heterogeneous distributed environment.

Input of the program implementing the method is PN T described in the format PNSF3 [10]. The input data are initially validated according to the corresponding document type definition (DTD). The master performs block decomposition of the net. After that, according to the obtained block structure, the master (Fig. 2) performs processor assignment by means of Algorithm 2a. The threads are initiated to control the slaves. Number of threads corresponds to number of the blocks ready to be processed, and number of the initiated threads corresponds to the number of slaves, each of which has some blocks to process, assigned by the module ProcessorAssignment. At each step a thread sends to the slave the initial markings of the corresponding block(s). A slave's task is verification of the given block and computing its terminal markings by means of Algorithm 2b. Result of the slave's work is sent back to the master, to the corresponding thread being in waiting mode. If the block is correct, the description of its terminal markings is sent to the main process of the master for further analysis of the net. If the block is incorrect, the program exits and further analysis is not performed.

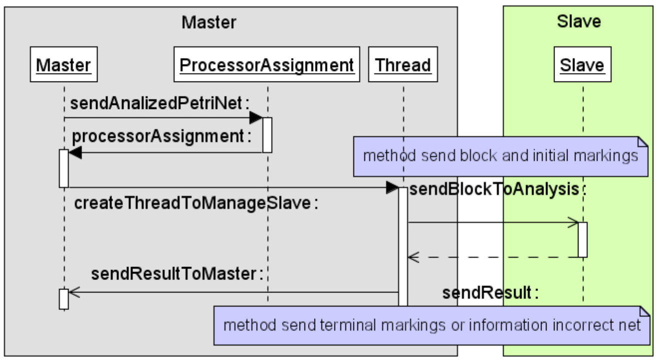

Fig.2. Operating model of Master and Slave as a sequence diagram.

JPVM environment serves as an intermediate layer between the master and the slaves. The master sends by its threads the messages to the slaves ordering to the JPVM demons of the slaves to initialize the corresponding analysis procedures. In the next JPVM messages the input data for those procedures are sent. The result of slave's work is a JPVM message with set of the terminal markings, if the block is correct, and the message that the block is incorrect otherwise.

V. Experimental Results

For experimental analysis of the presented approach, some tests have been performed. The system was tested at a separated local-area network based on 100Base-T technology. The network consisted of maximum 7 computers (which?). Parameters of the analyzed nets are given in Table 1.

TABLE 1

PARAMETERS OF THE NETS

	Places	Transitions
Net 1	128	118
Net 2	238	228
Net 3	360	349

Analysis time (in ms) is given in Table 2

TABLE 2

ANALYSIS TIME

Number of computers in the network	Net 1	Net 2	Net 3
1	2377	7277	12095
3	3202	6684	9092
5	3060	5893	8920
7	2626	5208	7578

The experimental results show that the method is efficient for big nets; for bigger nets it provides greater possibilities of time saving. This correlates to the results of the experiments on the sequential version of the analysis based on block decomposition [6]. For the small nets communication between the computers within the network noticeably decreases the gain.

VI. Conclusion

In this paper the method of analysis of big (with up to hundreds of places and transitions) Petri nets belonging to certain subclass is proposed. The method is intended for using in a distributed computing environment; it can be implemented on the basis of parallel virtual machines, FlexiNet or other systems of parallel network programming. The experiments show that network distribution can remarkably improve effectiveness of analysis of big Petri nets.

Of course, there are problems to be solved. One of them is evaluation of analysis time of a single block; formula (1) is far from being exact. Another problem is taking into account costs of data transfer in the network, important for analysis of some kinds of nets. These may be the directions of future work leading to develop a more efficient analysis system.

References

[1]. M. Adamski, Z. Skowroński, „Interpretowane sieci Petriego – model formalny w zintegrowanym projektowaniu mikroprocesorowych systemów sprzętowo-programowych," *PAK*, luty-marzec 2003, str.17-20.

[2]. A. Karatkevich, A. Zakrevskij, „Analysis of Petri Nets by means of Concurrent Simulation," in *Int. conference PARELEC'2002*, *IEEE*, Warsaw, Poland, Sept. 2002; pp. 87-91.

[3]. R. W. Lewis, "Programming Industrial Control Systems Using IEC 1131-3," *IEE*, London, 1995.

[4]. T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, pp.548-580, Apr. 1989.

[5]. C. L. McLeary, J. J. Tompson, D. G. Hill and others, "Partitioning and Scheduling Using Graph Decomposition"

[6]. A. Zakrevskij, A. Karatkevich and M. Adamski, „A Method of Analysis of Operational Petri Nets," in *ACS'2001,* Okt. 2001, Kluwer Academic Publishers, Boston, 2002.- pp. 449-460.

[7]. А. Д. Закревский, «Параллельные алгоритмы логического управления,» ИТК НАНБ, Минск, 1999.

[8]. A. J. Ferrari „JPVM: Network Parallel Computing in Java", Technical Report CS-97-29, Department of Computer Science University of Virginia, 1997

[9]. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam „JPVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing", Massachusetts Institute of Technology Press, 1994

[10]. A. Węgrzyn, P. Bubacz, „XML format for high-level Petri net" in *Advanced Computer Systems - ACS 2001*, Szczecin, Poland, 2001- Part 2, pp. 269-278