E11-2004-81

V. Kh. Khoromskaia[1,2]

PETRI NETS BASED MODELLING OF CONTROL FLOW FOR MEMORY-AID INTERACTIVE PROGRAMS IN TELEMEDICINE

[1]Joint Institute for Nuclear Research, Dubna, Russia
[2]Institute of Informatics, University of Leipzig, Germany

Хоромская В. Х. E11-2004-81

Моделирование на основе сетей Петри потока управления
в интерактивных программах поддержки памяти в телемедицине

Данная работа была инициирована проблемами моделирования интерактивных алгоритмов в рамках системы удаленной медицинской помощи для людей с нарушением памяти. Рассмотрено моделирование на базе сетей Петри (PN) потока управления в интерактивных программах поддержки памяти, предназначенных для использования в индивидуальных компьютерах и имеющих особые требования к надежности. Предложенная концепция моделирования алгоритмов позволяет разрабатывать программы с широкими возможностями для ежедневного планирования активной деятельности пользователей с учетом возможных ситуативных и временных ограничений.

Вначале мы строим модель PN для уже используемого простого алгоритма и анализируем его с помощью матрицы переходов и уравнения состояний. Затем мы строим граф PN для предлагаемого сложного алгоритма со взаимоисключающими возможными действиями и приводим вариант его анализа. Динамическое поведение этого алгоритма протестировано с помощью симулятора PN.

В работе показано, что применение PN-моделирования обеспечивает предсказуемое функционирование сложных интерактивных программ с разветвленной структурой и требованиями синхронизации.

Работа выполнена в Лаборатории информационных технологий ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна, 2004

Khoromskaia V. Kh. E11-2004-81

Petri Nets Based Modelling of Control Flow
for Memory-Aid Interactive Programs in Telemedicine

Petri Nets (PN) based modelling of the control flow for the interactive memory assistance programs designed for personal pocket computers and having special requirements for robustness is considered. The proposed concept allows one to elaborate the programs which can give users a variety of possibilities for a day-time planning in the presence of environmental and time restrictions.

First, a PN model for a known simple algorithm is constructed and analyzed using the corresponding state equations and incidence matrix. Then a PN graph for a complicated algorithm with overlapping actions and choice possibilities is designed, supplemented by an example of its analysis. Dynamic behaviour of this graph is tested by tracing of all possible paths of the flow of control using the PN simulator. It is shown that PN based modelling provides reliably predictable performance of interactive algorithms with branched structures and concurrency requirements.

The investigation has been performed at the Laboratory of Information Technologies, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna, 2004

INTRODUCTION

This work was initiated by the modelling and scheduling problems arising in the framework of a telemedicine project [7] aimed on a construction of computer assistance system for people with brain injuries. The problem consists in developing a scheduling paradigm for a human–computer interaction program, which should compensate memory disabilities of a customer while giving wide choice for a day-time planning in a presence of possible time limits for different actions*.

In existing approaches, the memory assisting programs were designed for managing simple user tasks which were restricted in time and space. Such programs possess rather transparent algorithmic structure, which can be modelled by an easily traced flow-chart. More complicated algorithms for a day-time schedule which can have time restrictions along with a possible changing of tasks and places require an appropriate modelling concept which could provide the construction of sophisticated real-time interactive programs. The latter should have a variety of choices and right responses and prompts for every choice with a free time planning from one side and with tracing of some important deadlines from the other side.

There are different approaches for modelling of the complex algorithms, with most widely used unified modelling language and flow charts. In our particular case we need possibilities for modelling of concurrency and interactivity, as well as the facilities for hierarchical subdivision. Here we choose the Petri Nets concept which provides a graphical and mathematical formalism for modelling, simulation and formal analysis of discrete event systems with concurrency which fairly suits for our problem of interest. This approach enables us not only to model the processes but also provides the facilities for tracing of all possible control flows in the designed structures, i.e., we can consider the dynamic behaviour of the modelled system. In this way, we obtain an integrated presentation of the software, environment properties and time constraints.

*In the framework of the project [7] a number of patients with memory disabilities are provided by pocket computers which have an access to a special server «guided» by a medical assistant. Every person has a separate plan for each day of the week. Individual plans are organized by medical assistants and are loaded into personal mobile computers at the beginning of each day. For critical cases a patient has access to a caregiver via mobile communication.

Application of Petri Nets (PN) for modelling of the interactive algorithms provides powerful tools for their analysis. In fact, there are well-developed means for mathematical analysis of such models. There is also a big choice of PN simulators which maintain analysis and/or animation, and in this way, provide verification of the performance of a chosen algorithm. Moreover, Petri Nets paradigm itself ensures the construction of a reliable algorithm since it clears out possible drawbacks of a concept already on the design level.

This paper presents an example which demonstrates, how tools and methods developed for the Petri Nets can be used for the development of secure and robust complex interactive algorithms.

1. BASICS OF PETRI NETS

Petri Net is a graph, consisting of two types of nodes, *places*, denoted by circles and *transitions*, denotes by bars. Directed arcs connect only different types of nodes: places with transitions, or transitions with places.

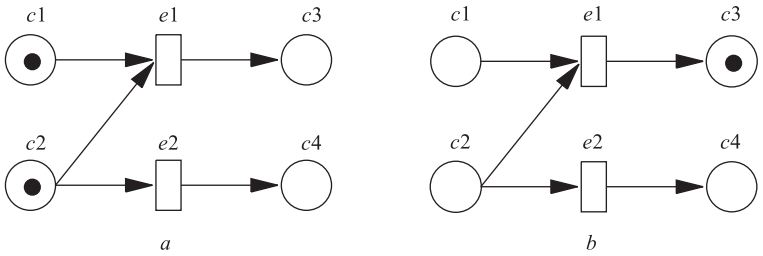

Fig. 1. Condition-event type of Petri Nets (*a*). The case (*b*) shows the graph after firing event $e1$

Places can contain marks, so-called *tokens*, which pass to the next places through transitions under certain conditions which will be discussed further. Tokens are denoted by black dots for condition-event Petri Nets (see Fig. 1), or by numbers, which correspond to the current number of tokens in a place for other types of Petri Nets (see Fig. 2). The place-transition Petri Net is a general type, while the condition-event type is a particular case of the place-transition graph when every place (condition) can contain only one token. However, this type of Petri Nets has some special properties [2].

Generally, the formal mathematical definition for the Petri Nets is the following:

- A net $\mathcal{N}$ is a 4-tuple $\mathcal{N} = (P, T, F, M_0)$, where
- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places,

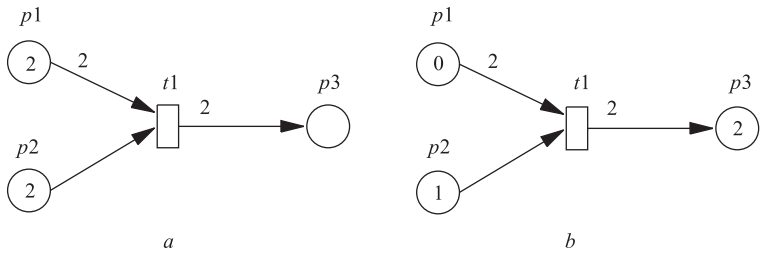

Fig. 2. Place-transition type of Petri Nets (*a*). The case (*b*) shows the graph after firing transition $t1$

- $T = \{t_1, t_2, \ldots, t_\ell\}$ is a finite set of transitions.
- For the above sets there holds $P \cap T = \emptyset$.
- The flow F is a set of arcs, $F \subseteq (P \times T) \cup (T \times P)$.
- M_0 is the *initial marking* of the Petri Net, that is a displacement of tokens in the net at the starting position. It shows, which places contain token at the beginning of the process.

Transitions can be *enabled* or disabled. A transition is enabled iff all places which have input arcs to this transition contain tokens. An enabled transition t_i can *fire*. It means, that it can be activated, so that tokens from each input place of t_i are transferred to each output place of this transition.

In place-transition Petri Nets some arcs get numbers, so-called *weights*, which tell how many tokens should be transferred through this arc. For example, the arc between $p1$ and $t1$ in Fig. 2 has weight 2. It means, that this arc is dedicated for transferring two tokens. And the input gate of $t1$ expects not less than two tokens from $p1$ to be enabled.

Examples of the performance of condition-event and place-transition Petri Nets are shown in Figs. 1 and 2, respectively. In Fig. 1 we observe the situation, when transition (event) $e1$ is fired first. Therefore, place (condition) $c3$ gets a token. In case if transition (event) $e2$ would be fired, place $c4$ would get a token, and so transition $e1$ would be disabled.

Figure 2 presents the example of a general place-transition graph with weighted arcs. The arc between $p1$ and $t1$ has weight 2, which is denoted in the graph by the corresponding number. If the arc has weight 1, as the one between $p2$ and $t1$, the number is omitted. In Fig. 2, *a* transition $t1$ is enabled (since it has enough tokens from both places), therefore it can fire. Fig. 2, *b* shows the graph after firing transition $t1$. Since the arc between $t1$ and $p3$ has weight 2, it moves two tokens to place $p3$. After firing the transition, one token is left in $p2$, since one firing of $t1$ needs one token from $p2$.

All interconnections of a given Petri Net can be described by an *incidence matrix*, which is given by

$$\mathbf{N} = \begin{cases} -1, & \text{if } (p,t) \in F \text{ and } (t,p) \notin F, \\ 1, & \text{if } (p,t) \notin F \text{ and } (t,p) \in F, \\ 0, & \text{if } (p,t) \notin F \text{ and } (t,p) \notin F. \end{cases} \quad (1)$$

It should be noted that the incidence matrix exists only for *pure* nets, i.e. nets without self-loops, consisting of one place and one transition (where p is both output and input place of t), because in this case the corresponding matrix elements are indefinite.

After firing a transition, the marking of the net changes from M_i to some M_j. A sequence of transitions $t_1, t_2, \ldots, t_k$ is called occurrence sequence, enabled at M, if there are markings $M_1, M_2, \ldots, M_k$, such that

$$M \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_k} M_k. \quad (2)$$

By tracing such sequences of markings it is possible to construct the *reachability graph*. The latter traces all variety of transitions and all reachable markings of the considered Petri Net. By analyzing the reachability graph one can reveal possible deadlocks, or can make a decision on boundedness of the system. It means, that the system contains a finite set of possible states (markings).

When analyzing a particular Petri Net, the main issues are reachability, boundedness and liveness of the given graph. The Petri Nets theory is presented in extensive literature [1–3,5,6], where the analysis and existence theorems for these properties are given. Here we do not include theoretical foundations of Petri Nets, we aim to show on basic examples how to construct and analyze these graphs.

Let us consider a Petri Net graph presented in Fig. 3 which models task processing by a processor unit. Here Petri Nets graph models the following processes: $t1$ denotes that a task is put into the queue, $p1$ — the task is waiting for the processor, $p2$ — the task is being processed, $p4$ — the processor is idle, $p3$ — the task is completed. For this graph we have the following flow:

$$F = \{t1p1, p1t2, p4t2, t2p2, p2t3, t3p4, t3p3, p3t4\}.$$

In Fig. 3 we have the initial marking of the graph $M_0 = (1, 0, 0, 1)$. It corresponds to the moment when there is a task waiting and the processor is idle (both places contain tokens). So next event which can happen, is that enabled transition $t2$ fires and place $p2$ gets a token. It means that the processor begins to compute the task. After the task is completed transition $t3$ fires and transfers tokens to places $p3$ and $p4$. After that the system has marking $M = (0, 0, 1, 1)$.

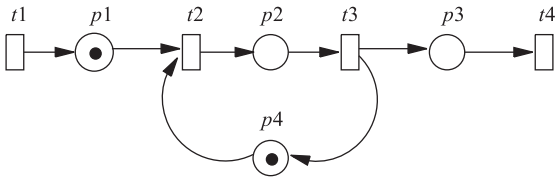

Fig. 3. Petri Nets graph for task processing

Incidence matrix (1) for the graph in Fig. 3 looks like

$$\mathbf{N}_{\text{Fig. 3}} = \begin{pmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & 0 \end{pmatrix}.$$

A well-known example of using condition-event Petri Nets for modelling the producer-consumer problem is shown in Fig. 4. The model consists of three parts, a producer, a buffer and a consumer. Producing and consuming parts have mutual dependence through the buffer states.

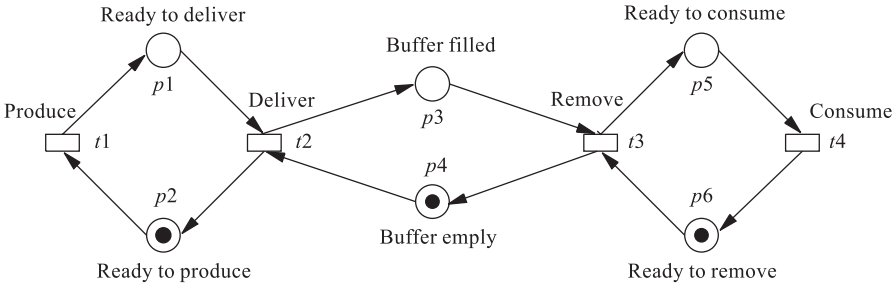

Fig. 4. Modelling of a producer-consumer problem

Here we begin with the initial marking $M_0 = (0, 1, 0, 1, 0, 1)$. In this position the only transition which can fire (or as we said, is enabled) is $t1$ (produce), while all other transitions are disabled at the marking M_0: $t2$ to be enabled requires tokens from both $p4$ and $p1$, and $t3$ to fire requires tokens from $p3$ and $p6$. Hence, after firing transition $t1$ we have the next marking $M = (1, 0, 0, 1, 0, 1)$. In this way, the token from place $p2$ is transferred to place $p1$. At this step transition $t2$ becomes enabled (both $p1$ and $p4$ contain a token) and can fire. It transfers tokens to both $p3$ and $p2$. The number of places which can obtain a token from a single enabled transition is defined only by a number of outgoing arcs but not

by the number of «incoming» tokens from the ingoing arcs. For the modelled process above steps mean that the product was produced and then delivered to the buffer. Next, we have the marking $M = (0, 1, 1, 0, 0, 1)$.

At this step we should note that firing of transition $t1$ is independent from the processes of the «consuming» part of the Petri Net. Performance of the «producing» part of the graph is influenced only by the state of the buffer. If the buffer is not empty, then the product cannot be «delivered» and further «production» stops. It is regulated by transition $t2$.

Next, in the «consuming» part, transition $t3$ is enabled, since both places which have incoming arcs to this transition contain a token. After firing $t3$, both $p4$ and $p5$ obtain a token. The «consuming» part of the net is also regulated by the state of the buffer from one side (transition $t3$), and by firing of independent transition $t4$ (consume) from the other side.

There are examples of modelling the producer-consumer problem for several consumer or/and several buffers, as well as with a number of tokens > 1.

One can find in the literature interesting examples of Petri Nets modelling for logistic processes, communication protocols [5], vending machines [3].

2. PETRI NET CONSTRUCTION FOR AN INTERACTIVE ALGORITHM

In Sections 3 and 4 we describe the main results of this paper. First, we show how to construct and analyze a Petri Net model for the flow-chart on an example of a medicine taking algorithm (see [7], page 11). According to presentation of the flow-chart elements in terms of places and transitions as described in [5], we generate a corresponding Petri Net (Fig. 5). The performance of the modelled algorithm can be investigated either by using state equations, or by building the reachability graph. Note, that in Fig. 5 transitions $t5$ and $t7$ are inserted for simulation reasons. The real algorithm has a start place at $p6$ and two exit places, $p5$ for normal exit and $p7$ for critical exit. This graph is safe, since the number of tokens in each place cannot exceed one.

To simplify further analysis we can reduce the Petri Net in Fig. 5 to the algorithm shown in Fig. 6. For this purpose we include the simple structures (having no branches) of the type $t - p - t$ into a single transition node t. In this way, we construct a compact graph for further analysis of the program. According to (1), the incidence matrix for this Petri Net is the following:

$$\mathbf{N}_{\mathrm{red.med}} = \begin{pmatrix} -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & -1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & -1 \end{pmatrix}. \quad (3)$$

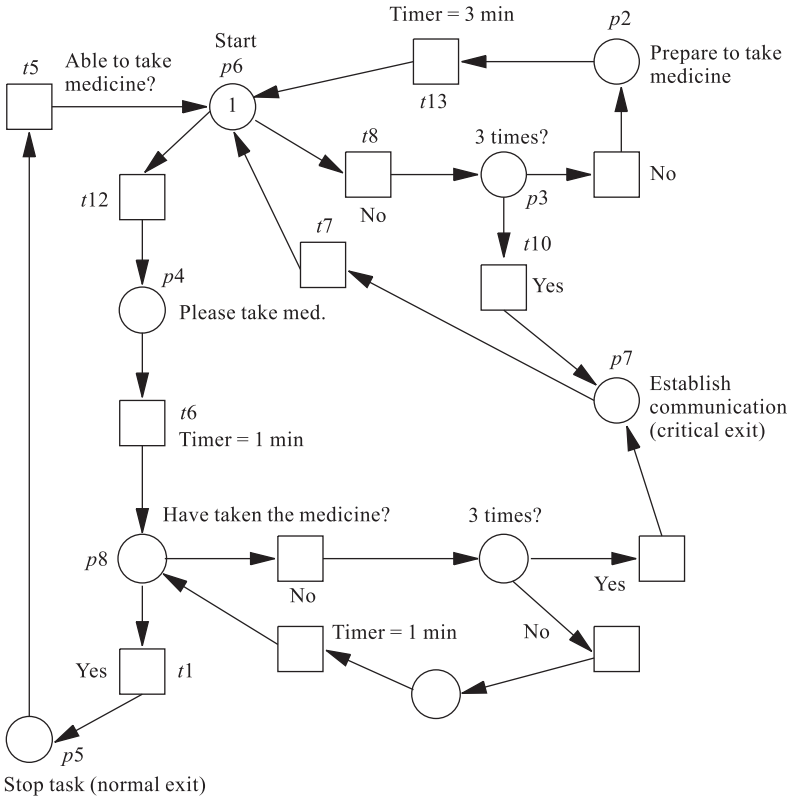

Fig. 5. Petri Nets graph for the medicine taking flow-chart. Transitions $t5$ and $t7$ are inserted for test simulation reasons

Every row of the incidence matrix corresponds to a definite place, every column — to a definite transition. For example, place $p5$ (fifth row from above) has two outgoing transitions $t7$ and $t8$, which are denoted by -1 and one ingoing transition $t6$, denoted by 1.

At the starting point we have the initial marking of the graph $M_0 = (1, 0, 0, 0, 0, 0)$. For any possible marking of the graph M we have equation [6]

$$M = M_0 + \mathbf{N} \cdot \mathbf{v}, \tag{4}$$

where $\mathbf{v}$ is the firing vector. For example, setting $M = (0, 0, 0, 0, 1, 0)$, we can find the firing vector by solving the system of equations

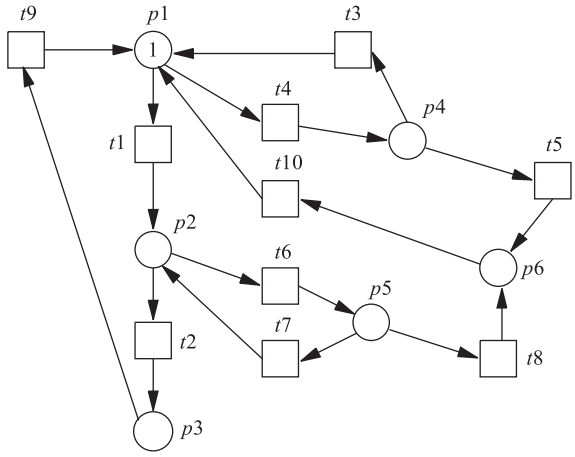

Fig. 6. Reduced graph for the medicine taking algorithm

$$\mathbf{N} \cdot \mathbf{v} = \begin{pmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

Hence, solving the system of equations

$$\begin{cases} -x_1 + x_3 - x_4 + x_9 + x_{10} = -1 \\ x_1 - x_2 - x_6 + x_7 = 0 \\ x_2 - x_9 = 0 \\ -x_3 + x_4 - x_5 = 0 \\ x_6 - x_7 - x_8 = 1 \\ x_5 + x_8 - x_{10} = 0, \end{cases}$$

we get the desired firing vector $\mathbf{v} = (1, 0, 0, 0, 0, 1, 0, 0, 0, 0)$, which, in fact, determines the sequence of transitions which should fire in order to reach the marking M from the initial marking M_0. Specifically, it means that transitions $t1$ and $t6$ should fire to reach a marking $M = (0, 0, 0, 0, 1, 0)$ from the initial marking $M_0 = (1, 0, 0, 0, 0, 0)$. In this way, it is possible to see if any marking in the designed graph is really reachable from a given state.

Another way of model analysis is based on the construction of the reachability graph. It is built by tracing the momentary possible markings of the system, as

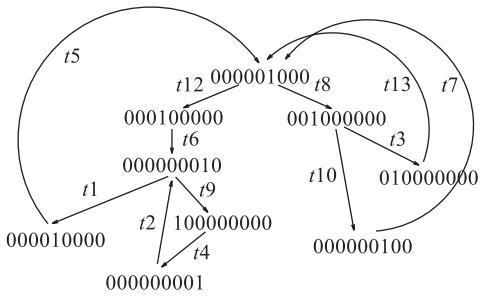

Fig. 7. Reachability graph for the medicine taking algorithm (in Fig. 5)

shown by (2), after each consequent firing, beginning from the starting position. The reachability graph for the initial Petri Net in Fig. 5 is shown in Fig. 7. Here, it is easy to see that the graph is bounded (it doesn't contain unbounded branches).

It should be noted that according to a definition [3], the graph presented in Fig. 5 belongs to a subclass of Petri Nets known as Finite-State Machines. It means that every transition in this net has exactly one incoming arc and exactly one outgoing arc.

In this way, any flow-chart can be modelled by a Petri Net, and on this basis we can derive its mathematical description, which, in turn, allows to analyze all possible and impossible states of the given algorithm. In the next section we construct a Petri Net graph for modelling parallel activities or concurrency.

3. THE ALGORITHM FOR OVERLAPPING ACTIONS WITH A CHOICE POSSIBILITY

In this section we propose the construction of algorithms for programming a day-time activity of a user which gives him a variety of possible activities according to his choice. These algorithms are modelled as Petri Net graphs which provide a «logical tracing» of the flow of control under the constraints coming from delays watching, displacement of the patient and a given variety of the day-task choices. The example of the PN graph constructed for three overlapping actions with choice possibilities is presented in Fig. 8. It describes the algorithm for programming choice between overlapping day-time tasks (actions) in a presence of time restrictions for every consequent action and taking into account possible variations in place location. It is possible to introduce more conditional restrictions, such as action priorities or some additional environment conditions. In Fig. 8 one can see how after starting place $p1 = 1$ (it corresponds to the initial marking $M_0 = (1, 0, \ldots, 0)$) the control flow is splitted into three

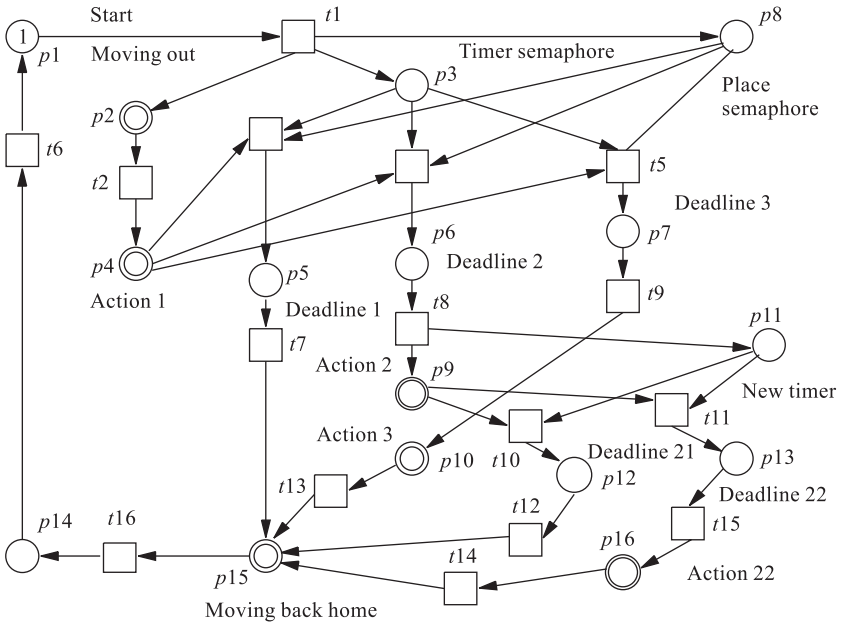

Fig. 8. Interactive algorithm with the two-stage overlapping actions using time and place semaphores

places $p2 = 1$, $p3 = 1$, $p8 = 1$, denoting the conditions for activity, time and space. The marking of the scheme becomes $M = (0, 1, 1, 0, \ldots, 1(p8), 0, \ldots, 0)$. These places have the meanings «moving out», «time semaphore» and «place semaphore». According to the time left after the first outdoor activity of a user denoted by «action 1», one of the three «deadlines» will be activated. These «deadlines» designate the fact that up to some *apriori* fixed time the user should return home, for example, to have some important meeting. The values of the «deadline» thresholds can be calculated by the program before starting the outdoor activity. And then, according to the time needed for the first action, program proposes the next route, or gives the possibility of available choice for the user. After that three branches of control flow are again incorporated into one by means of one of transitions $t6$, $t7$, $t8$ to form the next stage of activity depending on the previous conditions (on the choice of deadline route).

It is possible to introduce more semaphores, including some necessary preparations for the possible outdoor tasks. The graph may be more branched, but with the same logical structure.

The constructed graph has a hierarchical structure. The places drawn as double circles (for example, the places for «moving out»), like $p2$, $p4$ or $p9$ represent

the so-called virtual (macro)places. They denote that these places themselves contain subgraphs for implementing a certain activity, for example, moving from home to some place, or visiting a doctor. Most of macroplaces subgraphs have a structure analogous to the one corresponding to the medicine taking algorithm shown in Fig. 5. Generally, these simple graphs have normal and critical exits. Here we omit consideration of critical exits since their treatment can be easily implemented on the code level.

The activity of the program shown in Fig. 8 begins from place $p1$ («Start»). Then the following steps are activated:

• Firing of transition $t1$ activates places $p2$ («moving out of the house»), $p3$ («timer semaphore») and $p8$ («place semaphore»). As was noted, $p2$ is a macroplace, consisting of a graph itself.

• Firing of transition $t2$ activates place $p4$ («action 1»). This is also a macroplace, containing a simple subgraph for performing some predefined action.

• After that, according to the time past after action 1 is fulfilled, one of transitions $t3$, $t4$, or $t5$ can be enabled. The choice of firing depends on the predefined values of «deadline 1», «deadline 2» or «deadline 3» and on the desire of the patient. For example, if visiting a doctor took much time, then transition $t3$ is fired, corresponding to place «deadline 1», which leads to macroplace $p15$ «moving back home». If after «action 1» there is still some time left before some home activities, there can be a choice between firing $t4$ or $t5$, which leads to «action 2» or «action 3», according to the pre-defined day-plan of the user. The example of the choice of «deadline» for «action 2» is shown in Fig. 9.

• Transition $t8$ also activates new timer $p11$, which can be used for further choice of outdoor activities. In this case it is performing or not «action 22» according to the value of the timer and predefined values of «deadline 21» and «deadline 22».

Let us construct the incidence matrix for this scheme. According to (1) it looks like

$$\mathbf{N}_{\text{Fig.8}} = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 \end{pmatrix}. \qquad (5)$$

We can find analytically if some type of a marking is really reachable. The initial marking of the scheme is $M_0 = (1, 0, 0, \ldots, 0)$. We need to prove existence of

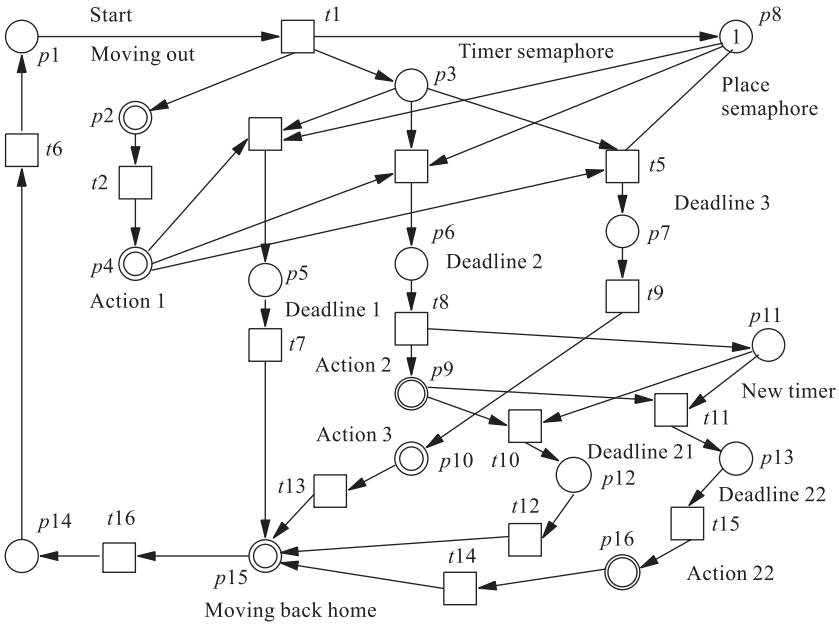

Fig. 9. Simulation step before a choice of the «deadline n» route. At this step, the tokens are distributed at places $p4$, $p3$ and $p8$ after firing the transitions $t1$ and $t2$ from the initial position $p1$ (Start)

some marking, say,

$$M = (0, \ldots, 0, 1(p9), 0, 1(p10), 0, \ldots, 0). \tag{6}$$

Solving the system of Eqs. (4) with the given M_0, M and incidence matrix $\mathbf{N}_{\text{Fig.8}}$ (5), we obtain the firing vector

$$\mathbf{v} = (1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0).$$

It corresponds to consequent firing of transitions $t1$, $t2$, $t4$ and $t8$, which bring the system to the state given by (6). Though with larger systems incidence matrices become rather cumbersome, they are easy for the treatment due to sparse structure.

Dynamic behaviour of the graph in Fig. 8 can be traced by the animation procedure («run simulations») of the Renew simulator [4]. One of the simulation steps is shown in Fig. 9. The Renew simulator provides step-wise tracing by manual activating of the transitions on the desired path or can produce automatic run-time simulations by «scanning» all enabled transitions one by one. Animation

in a run mode substitutes the analytical evaluation of the graph, since it traces all possible firing steps statistically. If there exists a place with unboundedness (a «trap») it would be seen after a few cycles of the run mode (the number of tokens in these places would be > 1).

CONCLUSIONS

There exists a number of references describing the Petri Nets based modelling for different fields of application, for example, simulations in internetworking, automatic production control, modelling of industrial concurrent processes, logistic management, etc. Here we have shown the application of Petri Nets for modelling of the control flow in the interactive memory assistance programs, which have special requirements for robustness. Presented examples of building and analyzing the PN models of interactive algorithms do not give an easy way for the problem solution. But they give a tool for verifying very branched program structures with concurrency, which is questionable when using flow charts or UML designs. For large complex algorithms with concurrency when mathematical analysis of a graph is cumbersome, it is possible to use PN simulators to estimate their performance. The proposed Petri Nets based modelling of the telemedicine tasks gives means for integrated presentation of environment properties, time constraints and for tracing of the flow of control in the designed structures.

In this work the Renew PN simulator was used [4], which provides good simulation equipment, as well as robust performance.

This work was partially supported by the SMWK Research Grant while V. Khoromskaia was working as a guest researcher at the Institute of Informatics, University of Leipzig, Germany.

Acknowledgements. I would like to thank Prof. Dr. K. Irmscher for the support of the current work and Dipl.-Inf. H. Schulze for the statement of the problem.

I am appreciative to Prof. Dr. S. Gerber for reading the manuscript and for useful comments on Section 2. I thank Dr. J. Waldmann for valuable remarks on the construction of the algorithm for overlapping actions.

I acknowledge Dr. E. Ayrjan for the support in publishing process.

REFERENCES

1. *Baumgarten B.* Petri-Netze. Grundlagen und Anwendungen. Spectrum, Acad. Verlag, 1996.
2. *Gerber S.* Petri-Netze. www.informatik.uni-leipzig.de/ theo/theo.html, 2000.
3. *Murata T.* Petri Nets: Properties, Analysis and Applications // Proc. of the IEEE. 77(4). 1989.

4. *Wienberg F., Kummer O., Duvigneau M.* Renew — User Guide. Release 1.6. University of Hamburg, Department for Informatics, 2002; www.renew.de.
5. *Peterson J. L.* Petri Net Theory and the Modelling of Systems. Prentice Hall, 1981.
6. Lectures on Petri Nets I: Basic Models. Advances in Petri Nets. Lecture Notes in Computer Science / Eds. W. Reisig, G. Rosenberg. Springer Verlag, 1998.
7. *Schulze H., Irmscher K.* A Mobile Distributed Telemedical System // Proc. of USM 2000 — Trends in Distributed Systems. Third International IFIP/GI Conference. Munich: Springer, 2000; www.memos-online.de/paper_dt.html.

Received on May 24, 2004.

Корректор *Т. Е. Попеко*

Подписано в печать 07.07.2004.
Формат 60 × 90/16. Бумага офсетная. Печать офсетная.
Усл. печ. л. 1,18. Уч.-изд. л. 1,67. Тираж 310. экз. Заказ № 54517.

Издательский отдел Объединенного института ядерных исследований
141980, г. Дубна, Московская обл., ул. Жолио-Кюри, 6.
E-mail: publish@pds.jinr.ru
www.jinr.ru/publish/