The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

TIMED PETRI NETS FOR SOFTWARE APPLICATIONS

Grzegorz ANDRZEJEWSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *g.andrzejewski@iie.pz.zgora.pl*

Abstract. *This paper presents a model of interpreted timed Petri nets in an abstract program environment called Virtual Decision System (VDS). There are described possibilities of software implementation with technical limitations of mentioned method. It shows a practical example with elements of deviation analysis. Possibilities of interpretation of dynamic parameters with special taking automatic decomposition algorithm into Hardware/Software Co-Design systems are described.*

Key Words. *Timed Petri Nets, Digital Microsystems, Software Applications*

1. INTRODUCTION

There exist a lot of ways for formal specification of digital control systems. The most of all (at present) are various modifications of Finite State Machine (e.g. CFSM, HCFSM), flow graphs (e.g. DFG, CDFG), hardware description languages (e.g. VHDL, Verilog) and Petri nets [7]. The latter have a special importance in designing process for the sake of rich formal verification apparatus and naturalness of concurrency aiding [5].
In designing process of control system very often a situation occurs in which specific after-effects of processes are strongly dependent on time. There is a simple example showing this problem on a simplified control system of initial washing in automatic washer (Fig.1).

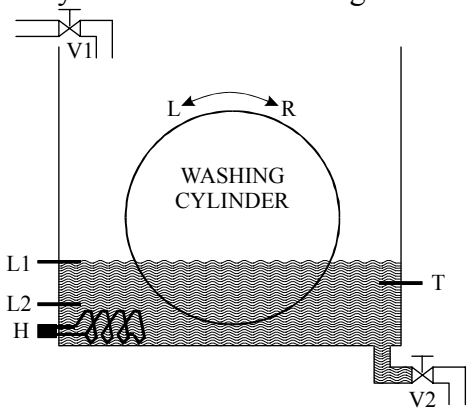


Fig. 1.An example of automatic washer control system

After turning on washing program valve V1 is opened and water is infused. Infusing process lasts to moment achieving of L1 level. In the same time after exceeding L2 level (total sinking of heater H) if the temperature is below required (TL1) heater system is turned on. After valve

V1 closing and water heating to temperature TL2 washing process is started in which the washing cylinder is turned alternate left and right for 10 sec. with 5 sec. break between. Keeping of temperature process is active for whole the washing cycle. The cycle is turned off after 280 sec. and cylinder is stopped, the heater is turned-off and the valve V2 is opened for water removing.
So, this problem is possible to describe by a hardware description language using *wait* and *after* instructions. But synthesis of them can give a lot of problems because these language constructions are not supported by synthesis tools. The other formal specification models (except Petri nets) don't allow a timing dependence in general.
In this article a new model of interpreted timed Petri nets for software applications is proposed, with its synthesis method. The method allows easy and cheap a practical implementation for shown dynamic system.

2. VIRTUAL DECISION SYSTEM

Program implementation of Petri net is possible in various ways. One is defining an abstract program environment making possible efficient implementation of control systems.
The proposed program environment is defined in terms of net places, names of input/output signals and program decision system called the Virtual Decision System VDS (Fig.2).

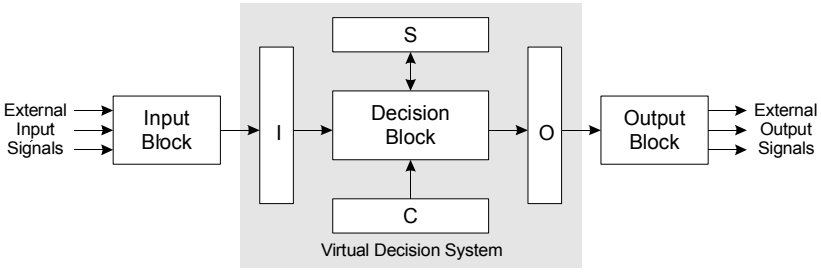


Fig. 2. Virtual Decision System

Where S – is a block storing information about system state, C – is a specification block, I – is a block storing the names of active input signals: $I = \{x \in X : x = 1\}$, O – is a block storing the names of active output signals: $O = \{y \in Y : y = 1\}$. It was broadly described in [8].

3. PROGRAM INTERPRETED PETRI NET

In [4] were introduced definitions describing program Petri net with most important kind of its: interpreted net, net with enabling and prohibit arcs. There is given a definition of program interpreted Petri net as a basis for further theoretical studies.

Program interpreted Petri net is a 5-tuple:

$$IPPN = (P, C, S_0, X, Y) \tag{1}$$

where: P is a non-empty set of names of net places, S_0 – is an initial net state, X is an input alphabet, Y is an output alphabet and C – is a specification structure composed of tables C^1 and C^2, such that: C^1 has $n \times 3$ size (n describes a number of decision rules) and C^1_{i1} i C^1_{i2} fields include sets of names of net places, which are correspondingly the conditions and the results of decision rule, C^1_{i3} field includes conditions in logical formulas form b(X) described on input alphabet; C^2 has $m \times 1$ size ($m = ||P||$) and each of fields includes set of names of output signals assigned to given place.

The conditions of rule *i* enabling:

$$\forall p \in C^1_{i1} : p \in S \tag{2-a}$$

$$b \in C^1_{i3} \Rightarrow b(X)=1 \tag{2-b}$$

The actions associated with rule *i* firing:

$$\forall p \in C^1_{i1} : S = S - p \quad (3\text{-a})$$

$$\forall p \in C^1_{i2} : S = S + p \quad (3\text{-b})$$

$$\forall (p \in C^1_{i1}) \forall (y \in C^2_p) : O = O - y \quad (3\text{-c})$$

$$\forall (p \in C^1_{i2}) \forall (y \in C^2_p) : O = O + y \quad (3\text{-d})$$

That means that from block S the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted. As well as from block O the names of output signals included in C^2 table's fields corresponding with places belonging to C^1_{i1} are removed and the names of output signals included in C^2 table's fields corresponding with places belonging to C^1_{i2} are inserted.

4. TIMED PETRI NETS

Timed Petri nets were introduced earlier in literature [1,6]. And yet the subject matter wasn't developed for the sake of implementation difficulties mainly. A new program model of timed Petri net is proposed in this paper, that allows an efficient implementation in microprocessor modules.

A general program model of interpreted timed Petri net is shown as an ordered 6-tuple:

$$TIPPN = (P, C, S_0, X, Y, T) \quad (4)$$

where: P, S_0, X, Y are defined just like (1), C is a specification block defined according to a kind of described net, and T is discrete scale of time. The differences will be stressed also in conditions of rules enabling and actions connected with their firing.

There is modified block S, keeping information about system state. For timed nets it can be a structure composed of following elements:

$S_1 = \{p: p \in P\}$ – includes names of places currently marked, being outside a keeping state,

$S^1_2 = \{p: p \in P\}$ – includes names of places currently marked, being inside a keeping state,

$S^2_2 = \{t: t \in T\}$ – includes numbers assigned to discrete scale of time and describing state of timing actions in accordance with the places in S^1_2,

$S^1_3 = \{p: p \in P\}$ – includes names of places prepared for marking after closing adequate timing actions assigned to given rules firing,

$S^2_3 = \{t: t \in T\}$ – includes numbers assigned to discrete scale of time and describing state of timing actions in accordance with the places in S^1_3.

4.1. Program timed net of P-type

The time parameters are assigned to places in the nets of P-type and they are called keeping times. The keeping times can be perceived as times of staying marker in place *p* what is shown in Fig.3-a) (along with practical interpretation).

The condition of transition T2 enabling is a time (2 sec.) passing since moment of introducing marker into place P1. In practice this situation is interpreted as starting external timing action by signal c1. The end of action is indicated by set signal c2 conditioning transition T2.

A theoretical model is consistent with (4). Specification block C has differences to (1):

table C^2 has m×2 size, into $C^2_1(p)$ are included names of output signals assigned to place *p*, and into $C^2_2(p)$ are included numbers assigned to discrete scale of time T, describing the keeping time of marker in place *p*.

The condition of rule *i* enabling (just like 2-b) and:

$$\forall p \in C_{i1}{}^1 : p \in S_1 \quad (5)$$

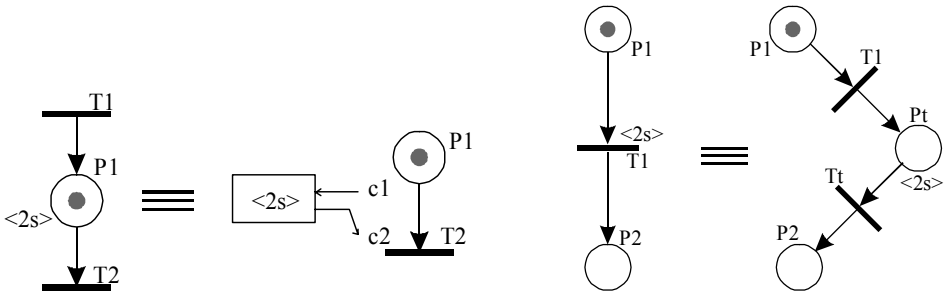
The actions associated with rule *i* firing:

$$\forall p \in C^1_{i1} : S_1 = S_1 - p \quad (6\text{-a})$$

$$\forall p \in C^1_{i2} : C^2_2(p) = 0 \Rightarrow S_1 = S_1 + p \quad (6\text{-b})$$

$$\forall p \in C^1_{i2} : C^2_2(p) \neq 0 \Rightarrow S^1_2 = S^1_2 + p \text{ i } S^2_2 = S^2_2 + C^2_2(p) \quad (6\text{-c})$$

That means that from block S_1 the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted to S_1 if their keeping time equals zero ($C^2_2(p) = 0$), or to S^1_2 if not. Simultaneously a variable of timing action is added to S^2_2 (with initialization by $C^2_2(p)$ value).



a) *b)*

Fig. 3.An example of a) P-type net, b) T-type net

4.2. Program timed net of T-type

The time parameters are assigned to transitions in the nets of T-type and they are called execution times. The execution times can be perceived as times since moment of removing markers from input places to moment of inserting marker to output places given transition *t*. The situation can be interpreted with ideas P-type net introduced in 4.1. During transition T1 execution (Fig.3-b)) marker is moved to auxiliary place Pt. End of this execution (equivalent to keeping time of place Pt) allows auxiliary transition Tt execution. In consequence the marker is moved to output place P2.

A theoretical model is consistent with (4). Specification block C has differences to (1):
table C^1 has n×4 size, fields C^1_{i1}, C^1_{i2}, and C^1_{i3} just like (1), and into C^1_{i4} are included numbers assigned to discrete scale of time T, describing execution time of transition *t*.

The condition of rule *i* enabling just like (5). The actions associated with rule *i* firing just like (6-a) and:

$$C^1_{i4} = 0 \Rightarrow \forall p \in C^1_{i2} : S_1 = S_1 + p \quad (7\text{-a})$$

$$C^1_{i4} \neq 0 \Rightarrow \forall p \in C^1_{i2} : S^1_3 = S^1_3 + p \text{ i } S^2_3 = S^2_3 + C^1_{i4} \quad (7\text{-b})$$

That means that from block S_1 the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted to S_1 if execution time of rule equals zero ($C^1_{i4} = 0$), or to S^1_3 if not. Simultaneously a variable of timing action is added to S^2_3 (with initialization by C^1_{i4} value).

4.3. Program timed net of PT-type

Timed Petri net PT-type is a superposition of nets defined in 4.1 and 4.2. There are the time parameters assigned with both places and transitions. A theoretical model is consistent with (4). Specification block C is composed of two tables: C^1 (just like in 4.2) and C^2 (just like in 4.1).

The condition of rule *i* enabling just like (5). The actions associated with rule *i* firing just like (6-a) and:

$$C^1_{i4} \neq 0 \Rightarrow \forall p \in C^1_{i2} : S^1_2 = S^1_2 + p \text{ i } S^2_2 = S^2_2 + C^1_{i4} \quad (8\text{-a})$$

$$C^1_{i4} = 0 \Rightarrow \forall p \in C^1_{i2} : C^2_2(p) = 0 \Rightarrow S_1 = S_1 + p \quad (8\text{-b})$$

$$C^1_{i4} = 0 \Rightarrow \forall p \in C^1_{i2} : C^2_2(p) \neq 0 \Rightarrow S^1_3 = S^1_3 + p \text{ i } S^2_3 = S^2_3 + C^2_2(p) \quad (8\text{-c})$$

That means that from block S_1 the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted to S_1 if execution time of rule equals zero ($C^1_{i4} = 0$) and keeping time equals zero ($C^2_2(p) = 0$), or to S^1_2 if execution time of rule doesn't equal zero ($C^1_{i4} \neq 0$), or to S^1_3 if execution time of rule equals zero ($C^1_{i4} = 0$) and keeping time doesn't equal zero ($C^2_2(p) \neq 0$). The variables of timing action are adding to S^2_2 and S^2_3 simultaneously with (8-a) and (8-c).

5. RESULTS AND APPLICATION

So the defined environment and program implementation model of interpreted Petri net enables simple implementation with any high level language. Specification of blocks C and S can be declared as structures composed by required number of tables; blocks I and O – as global variables. Decision block can be declared as a system of functions operating on mentioned blocks. It was broadly depicted in [2,4].

This paper presents only methodology of initializing and performing of timing actions in simple systems with industrial standard microcontrollers.

The idea of executing timing actions relies on using interrupt system in microprocessor module. In general it must generate an interrupt with given constant frequency (e.g. 1 kHz). It is possible to realize by using a generator connected to external interrupt pin or by using an internal timer/counter (more recommended). The overflow that timer is indicated by calling right interrupt. The interrupt handling procedure is used to decrement of auxiliary registers storing information about actual state of timing actions.

At the initialization moment of timing action there is granted suitable variable with initial value $t_p = t_a * f_i$, where t_a is a required time of action, f_i is a frequency of calling interrupt *int*. The value of t_p is taken from right fields of specification block C (C^1_{i4} and $C^2_2(p)$).

During interrupt handling procedure microprocessor performs decrement operations all variables of timing actions. If any variable equals zero then it is taken operation assigned to its ending and in next step it is removed from variables list:

$$\forall (p \in S^1_2)\ S^2_2 = 0 \Rightarrow S^1_2 = S^1_2 - p \text{ and } S^2_2 = S^2_2 - S^2_2(p) \text{ and } S_1 = S_1 + p \quad (9\text{-a})$$

$$\forall (p \in S^1_3)\ S^2_3 = 0 \Rightarrow S^1_3 = S^1_3 - p \text{ and } S^2_3 = S^2_3 - S^2_3(p) \text{ and } S_1 = S_1 + p \quad (9\text{-b})$$

Contents of block O is updated according to contents of blocks S_1 oraz S^1_2:

$$\forall [p \notin (S_1 \cup S^1_2)]\ \forall [y \in C^2_1(p)]\ O = O - y \quad (10\text{-a})$$

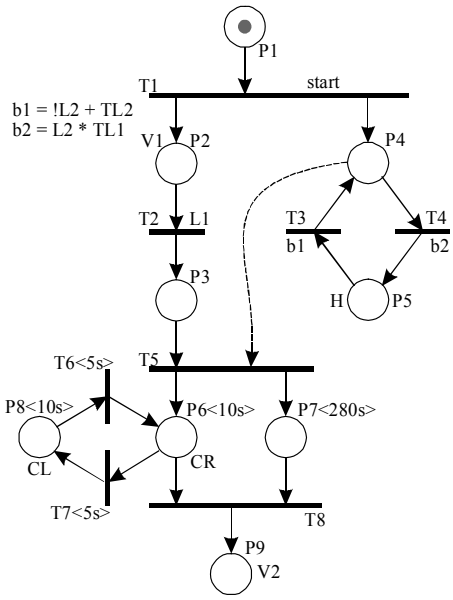
$$\forall [p \in (S_1 \cup S^1_2)]\ \forall [y \in C^2_1(p)]\ O = O + y \quad (10\text{-b})$$

A precision of timing action is dependent on a lot of factors in proposed system. The most important are: inaccuracy of generating interrupt system, delay of calling interrupt procedure, time of interrupt handling, number of active timing actions, time of updating state block S, reaction time of decision block and time of function I/O handling.

Suppose sufficient precision generating interrupt system there we can omit the deviation assigned to it. The other factors are strongly dependent among other things on used processor, its clock speed and quality of executable program code generated by given compiler.

In general the deviation of timing action is linear to number of decision rules and it's contained in given range. The range is possible to calculating in analysis process. A specific value of this deviation (in the range of course) for given rules or places is dependent on location the rules in specification block C. This is a result of sequential examining block C by decision block in permitted rules searching.

A net modeling that system and an obtained results are presented in Fig.4.



Parameter	Result
Total program memory occupation	968 B
Data memory occupation	148 B
Occupation of specification block C	100 B
Deviation of timing action (min.)	44 μs
Deviation of timing action (max.)	80 μs
Reaction time (min.)	0.6 ms
Reaction time (max.)	2,2 ms

Fig. 4. Fragment of the net modelling washer controller with the results

As a test system was adopted single-chip microcontroller PCF 80C51HB-3 with 12MHz clock. Program was compiled with μVision 2 tool (Keil Software).

6. SUMMARY

Application of the timed model Petri nets not only can effectively help on description level for reactive system strongly time depended but it can be used in system decomposition automation too (in Hardware/Software Co-Design sense [3]).
Further works and researches are to steer on studying generalized model for complicated hierarchical nets. We are going to consider not only simplifications of implementation for nets with complicated topology but also implementation of nets in distracted systems.

This work was supported by KBN grant: 7 T11C 010 20.

REFERENCES

[1] Adamski M.: *Digital systems design by formal transformation of specification*, Prep.35 Int. Wissen. Koll., TH Ilmenau, Germany, 1990, Heft 3, pp.62-65

[2] Andrzejewski G.: *Parallel controllers design with program model of interpreted Petri net*, OWD'2000, Istebna-Zaolzie, 22-25.10.2000, pp.63-68 (in Polish)

[3] Andrzejewski G.: *System decomposition in hardware/software co-design*, RUC'2001, Szczecin 7-8.05. 2001, pp.117-124 (in Polish)

[4] Andrzejewski G.: *Program model of Petri net*, accepted for publication in conference proceedings CAD DD'2001, Minsk 14-16.11.2001, Belarus

[5] Banaszak Z., Kuś J., Adamski M.: *Petri nets, Modeling, Control and Synthesis of Discrete Systems*, printed series of course lectures WSI in Zielona Góra, 1993 (in Polish)

[6] Bolton M.P.J.: *Digital systems design with programmable logic*, Addison Wesley Publ. Company, Wakingham, 1990

[7] Gajski D.D., Vahid F., Narayan S., Gong J.: *Specification and Design of Embedded Systems*, Prentice Hall, Englewood Cliffs, NJ, 1994

[8] F.Wagner: *The Virtual State Machine: Executable Control Flow Specification*, Rosa Fischer-Löw Verlag, 1994,ISBN3-929465-04-3