ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 *2011. No. 2(108)*
ELEKTRONIKA IR ELEKTROTECHNIKA

AUTOMATION, ROBOTICS
T 125
AUTOMATIZAVIMAS, ROBOTECHNIKA

Simulation using Interruptions

S. Bartkevicius
Department of Electrical Engineering, Kaunas University of Technology,
Studentų str. 48-230, LT-51367 Kaunas, Lithuania, phone: +370 37 300253, e-mail: stanislovas.bartkevicius@ktu.lt
K. Sarkauskas
Department of Control Engineering, Kaunas University of Technology,
Studentų str. 48-107, LT-51367 Kaunas, Lithuania, phone: +370 61025585, e-mail: kastytis.sarkauskas@ktu.lt
A. Vilkauskas
Mechatronics Centre for Research, Studies and Information, Kaunas University of Technology,
Kestučio str. 27, LT-44312 Kaunas, Lithuania, phone +370 61024708, e-mail: andrius.vilkauskas@ktu.lt

Introduction

There are known two methods of organization of control programs in discrete control systems – polling of devices or subsystems and interruption service. The first one is simple to program, but is used in relatively not complicated systems, because requires many resources of a processor for continuous polling of devices. The second one requires of a processor attention only if a service is needed. This necessity calls an interruption of a running program to make a service. This method is more efficient in sense of usage of resources, because continuous checking of peripherals devices is not required.

It is helpfully to create and check control programs and methods using simulation systems. The authors have built and use program package CENTAURUS CPN [1] for this purpose. The aim of this paper is to show, how control systems with interruption service may be simulated by means of this package. The necessity of design of such means was called by simulation of traffic control in a ring crossing. The simulation, using polling, becomes very slow and not effective, if number of cars increases. For example, when 21 car is in such crossing with 4 ways, 420 (21*20) conditions must be checked to avoid collisions on entering the ring, and the same number of conditions is to be checked to bypass overrides from the rear [2], when a car moves around in the ring. Thus, total number of checks to be made on each step of simulation – 840 and model becomes not functional

Solution of the problem of redundant checks

The checking's, of conflict conditions on each step of the Petri net seems to be redundant, if organize control with interruptions. It is necessary to show parameter (expressions), which may have any transition the net realized by the means of CENTARUS CPN, before that. Any transition may be delayed, except of conventional guard expression. Delay time is defined by „delay expression". Usage of colored Petri nets as simulation tool of control systems requires, as shown in [3], global variables. It is a bit dangerous, but some special limitations, such as prohibited change of value of a global variable until a current step of Petri net is not complete, let to avoid hidden links or cyclic references. Some extensions are made. A transition can have additional attribute – „finish expression". Values of global can be changed only by finish expressions, which all are calculated after a current Petri net step is finished. A change of a global variable value may cause branching conditions of the net. On other hand, it may be a reason to make an interruption.

A transition, in a conventional Petri net, can be fired only if all values of expressions of input links are found as tokens in appropriate places, connected to these links. Additional boolean expression (guard expression) - must be satisfied too, if declared. Fired transition causes calculation of expressions of output links, i. e. generation of output tokens. Therefore, a transition without input links never can fire.

An interruption of running program on one processor controller means, that execution of the program is temporarily stopped and a procedure of the interruption service is executed until finished. The interrupted program is continued from the brake point after that. Petri nets suppose parallel firing of transitions; therefore let us treat an interruption of running program, simulated by Petri net, as firing of a transition on a special conditions, despite of presence of input links.

These special conditions are three:

1. Value of a global variable is changed;
2. Special function, causing interruption, is present in the guard expression of a transition;

3. This boolean function returns value „true“.

The special function to cause an interruption is called *Event(fun)*. The parameter *fun* is parameter less function defined by user in declarations. The function *fun* operates only with constants or global variables because of absence of input parameters. It may be concluded, that *there is no need to execute this function, in order to check if an interruption arise, if no one of global variables is changed.* Furthermore, it is possible in the phase of compilation of declarations to define global variable or group of them, change of that can cause an interruption. The fact of change of a global variable can be easily checked when finish expressions are calculated.

The algorithm of one step of simulation process written in pseudo-code is such:

for *all transitions* **do**
  if *input links exist* **then**
    Calculate values of expressions of input links;
    Check, if tokens equivalent for these values exists in places, to which input chords are connected;
    if *all input expressions satisfied* **then**
      if *value of guard expression is* **true** **then**
        {*Fire this transition*}
        Calculate values of expressions of output links;
        if *finish expression defined* **then**
          Evaluate it and put the result into temporary store
    else if *a global value has changed before* **then**
      if *guard expression contains function event and this function returns* **true** **then**
        {*Fire this transition. An interruption rises*}
        Calculate values of expressions of output links;
        if *finish expression defined* **then**
          Evaluate it and put the result into temporary store;
  Trim up tokens in places – remove tokens used by input links and put tokens generated by output links.
  Remember new values of global variables generated by finish expressions and stored in temporary store.

The underlined code is added to simulate interruptions.

The example shown in Fig. 1 explains the written above. The function *go,* defined in declarations returns result „true“ then value of global boolean variable *R1* becomes „true“. This function is included into guard expression of transition **6.** Value of *R1*can be changed in finish expression of delayed transition **2** by function *Zap*, which realizes assignment of value „true“ to global value *R1*.The transition **6** fires only after that, when event *R1= true* occurs.

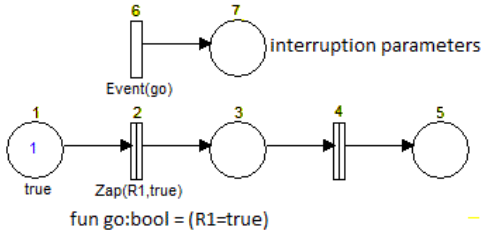

Fig. 1. Model with interruption. **1–5** simple model, **6–7** interruption control

This event originates at the end of the step of the net, on which delayed transition **2** was fired, delay time has finished and finish expression of this transition is calculated, in other words, when all three, before mentioned, conditions are fulfilled.

Simulation with interruptions

The aim of this paragraph is to show differences in modeling and simulation using polling and interruptions. The traffic simulation in a ring crossing was used only as a test task, show that new modeling system is functioning.

The principles of building Petri net model of the traffic are described in [3, 6] and a crossing is shown in Fig. 2. A riding car must keep secure distant from a car before, therefore continuous control of distance is necessary. On other hand, a car entering the ring must let a car riding around the ring, if distant between cars is not sufficient [4, 5, 7].

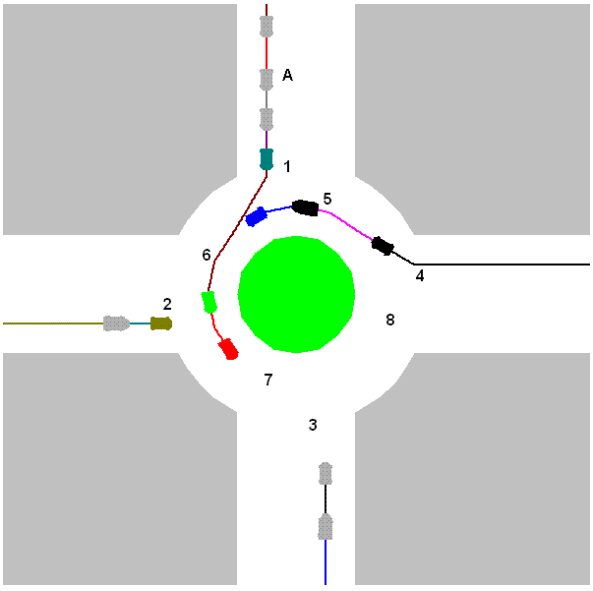

Fig. 2. Ring crossing with riding cars. *1–4* the zones of entering the ring, *5–8* zones „ring is busy“, **A** – riding car

It is evident, that such task is asynchronous and stochastic, because appearance of a car near the crossing is random.

A car *A* can enter and leave the crossing by any of four ways (Fig. 3), so it seems reasonable to split the task

of riding throughout the crossing into several smaller tasks:

1. Check, on each simulation step, if a car reached a zone (1–4) of entering the ring;
2. Check, if a zone „ring is busy“(5–8) is free, because the presence of a car in these zones prohibit entering the ring. It means that checking of all cars, if they are not in a zone of 5–8 is necessary. The zones 1–4 and 5–8 are coherent in pairs – zone 1 with 5, 2 with 6 and so on;
3. Check, on each simulation step, if each car do not overtake another car to avoid collision.

The fragment of the net (Fig. 3) shows, how a „riding“ of a car is realized.

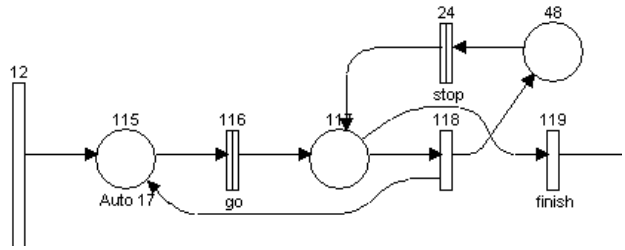

Fig. 3. The net fragment, representing the riding of a car. Names of transitions and actions, that are simulated by firing of them: *go* – car is riding, *finish* – car is out of simulation field, *118* – car can move or must stop, *stop* – car is waiting during one simulation step

The delayed transition 116 simulates the movement of a car in one simulation step, in time and distance.

The new position of the car, dot type token, is generated by output chord of this transition and remembered if the place 117. The car, if the new position of it is out of the simulation field, is „sent“ into the part of the model, which organizes movement of new cars, to be used repeatedly. The token, representing car is sent, throughout the transition 118, into place 115, if car can move the next step and into 48, if car must stop for a while, to avoid a possible collision. All before mentioned checks are done in the environment of the transition 118. Checking, if a car reached the crossing, is done at the 118 transition too.

The main amount of time necessary to simulate riding of a car is wasted for checks associated with the transition 118. The model with 21 car becomes very slow and not useful practically.

The same crossing (Fig. 2) can be simulated using proposed interruptions. The continuous, from step to step, checking conditions of interruptions is necessary now only. Many of conditions formulated above can be checked when an interruption rises.

There are three kinds of interruptions in the model:

1. Interruption occurs, when a car enters one of zones 5–8. The fact, which car did that, is not important at all;
2. A car entered into one of zones 1–4. It is important, which car did it and into which zone, here. The service program must check, if a coherent zone, 5–8, is free. The car can enter the ring if „ring is busy“ zone free and stop, if busy;
3. A car overtakes another car, moving before it. Interruption rises, if distance between both cars becomes less then secure one. The service program stops the car.

Services of these interruptions are nested, e.c. the interruption of the first kind generates conditions of the branching on service of a interruption of the second type.

The structure of the fragment of the net, where services of the interruptions of the first kind are organized, is shown in Fig. 4. Transitions 614 – 617 fires, when interruptions, called by entering of a car into zone 5 – 8, respectively, near the south, east, north and west, ways entering the ring rises. Therefore, these four fragments simulate service programs of each interruption.

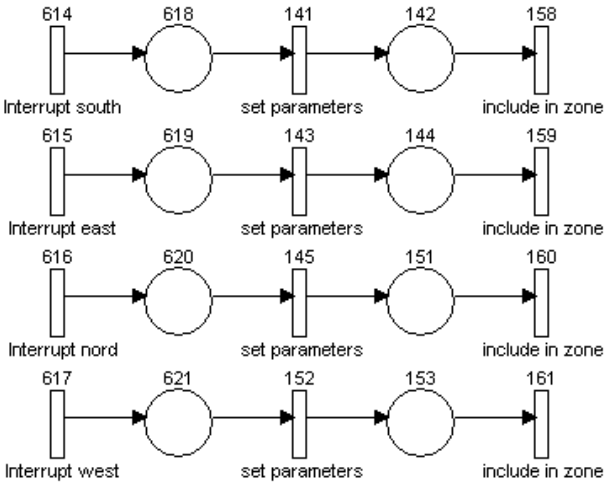

Fig. 4. Subnet realizing interruptions, when a car enters one of zones 5 – 8, service of the interruptions is organized here too

Transitions 141–152 have finish expressions to make changes of global variables, which are used in user defined functions setting necessity of an interruption. Transitions 158–161 check and fix new values of global variables – symptoms of business of zones 5–8. Therefore, it is not necessary organize checking of this condition for all cars on each simulation step. It is one of reasons, why speed of simulation rises.

The fragment of the net (Fig. 5) shows, how a „riding“ of a car is realized with interruptions.

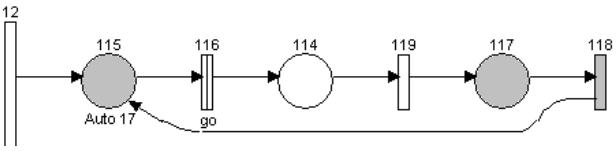

Fig. 5. The net fragment, simulating riding of a car step by step: *go* – car is riding, *118* – subnet, where checking of conditions of interruptions is modeled

Firing of transition 116 is equivalent to movement of a car by one step. All checks of before mentioned interruption conditions are realized in the subnet, represented by transition 118.

The structure of the subnet 118 is shown in Fig. 6. The fragment in this picture illustrates simulation of a car riding throughout the crossing (Fig. 2) with interruptions. Transition 760 fires, when a car moves, on current simulation step, without any conflict. Transition 762 fires, when a

car leaves the area of the crossing. A conflict causes firing of the transition 764 and it is equivalent to stop of a car for a time interval, which is realized as delay of the transition 765.

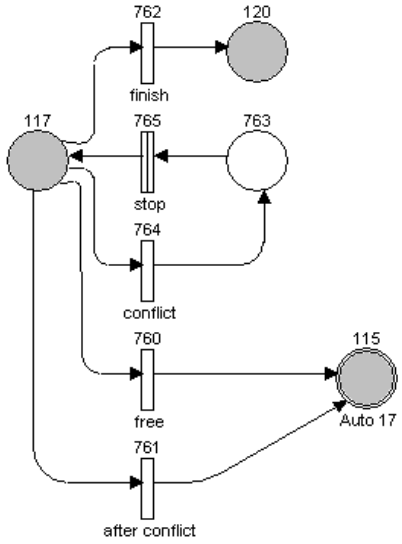

Fig. 6. The subnet fragment, representing the riding of a car, when interruptions are used. Names of transitions and actions, that are simulated by firing of them: *Free* – car can ride, because there is no obstacles, *conflict* – an obstacle fixed, *stop* – car is waiting during one simulation step, after conflict – permission to ride is granted for a car, *finish* – car is out of simulation field

The transition 761 fires after conflict is solved and a car may continue movement under control of the transition 760.

It can be noted, that there is no need to check conflict conditions on each step of simulation. It lets significantly increase the simulation speed.

Conclusions

Simulation of control systems with large amount of interacting objects requires a powerful computer, if process is realized using polling. The number of conditions to check becomes huge and these checking must be done on each step of simulation. Simulation process becomes slow and, practically, not useful.

The new simulation technique and program package using interruptions generated by the model itself are proposed. This technique, applied to simulation of traffic throughout a ring crossing, allows significantly simplify the model structure and make the simulation process appreciably faster.

References

1. **Bartkevičius S., Šarkauskas K**. Spalvoti Petri tinklai. Programinis paketas CENTAURUS CPN. – Kaunas: Technologija, 2008. – 59 p.
2. **Simutis R., Narvydas G.** Autonomous mobile robot control using IF–THEN rules and genetic algorithm. // Information technology and control, 2008. – Vol. 37. – No. 3. – P. 193–197.
3. **Baranauskas V., Bartkevičius S., Dervinienė A., Šarkauskas K.** Rationalization of the Path Search Algorithm. // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 5(101). – P. 72–82.
4. **Gning A., Mihaylova L., Boel R.** An interval compositional vehicular traffi c model for real–time applications // IEEE Intelligent Vehicles Symposium. – Eindhoven, Netherlands, 2008. – P. 494–499.
5. **Baranauskas V., Bartkevičius S., Dervinienė A., Šarkauskas K.** Influence of dimensions of a mobile robot into navigation trace // Proceedings of International Conference Electrical and Control Technologies'2010. – Kaunas: Technologija. 2010. – P. 47–50.
6. **Baranauskas V., Bartkevičius S., Šarkauskas K.** Creation of vector marks for robot navigation // Electronics and Electrical Engineering. – Kaunas: Technologija, 2008. – No. 4(84). – P. 27–30.
7. **Tautkus A., Bazaras Ž.** Modelling and Investigation of Car Collisions // Transport. –Vilnius, 2007. – No. 22(4). – P. 279–283.

Received 2010 10 21

S. Bartkevicius, K. Sarkauskas, A. Vilkauskas. Simulation using Interruptions // Electronics and Electrical Engineering. – Kaunas: Technologija, 2011. – No. 2(108). – P. 81–84.

Simulation of control systems with large amount of interacting objects requires a powerful computer, because traditionally systems with polling are used. The number of conditions to check on each step of simulations becomes so large, that models become unfunctional. The new simulation technique, using program interruptions, which are organized by the model itself is proposed. Replace of polling by interruptions lets radically increase the speed of simulation and simplify the model structure. Ill. 6, bibl. 7 (in English; abstracts in English and Lithuanian).

S. Bartkevičius, K. Šarkauskas, A. Vilkauskas. Valdymo sistemos modeliavimas taikant pertraukimų principą // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2011. – Nr. 2(108). – P. 81–84.

Sistemoms su daugeliu sąveikaujančių objektų modeliuoti reikia labai spartaus kompiuterio, nes įprastinis modeliavimas atliekamas nuosekliu apklausos principu. Tokioje sistemoje susidaro tiek daug sąlygų, tikrintinų kiekviename imitacijos žingsnyje, kad modeliai tampa nebefunkcionalūs. Siūlomas naujas modeliavimo metodas, kai pats modelis gali organizuoti pertraukimus atsisakydamas programavimo, paremto nuoseklios apklausos principu. Apklausos principo pakeitimas pertraukimo metodu gerokai padidino modeliavimo greitį ir supaprastino pačią modelio schemą. Il. 6, bibl. 7 (anglų kalba; santraukos anglų ir lietuvių k.).