Automatic Linear Robot Control Synthesis Using Genetic Programming

Tiberiu S. Letia and Octavian Cuibus

Technical University of Cluj Napoca, Cluj Napoca, RO 400114, Romania
Tiberiu.Letia@aut.utcluj.ro, ocuibus@yahoo.com

Abstract. An automatic controller synthesis method for a single axe linear robot is considered. The robot motions are modeled by a Delay Time Petri Net (DTPN). The search refers to automatically finding a controller modeled by a Time Petri Net (TPN) that fulfills some specified requirements. The controller model is synthesized using a Genetic Programming (GP) method. The mapping between TPN model and the tree representation of individual genotypes is performed using a formal language named here TPNL (Time Petri Net based Language). This language is suited for formal description of the controller behavior traits like sequential, concurrent, selection, loop or input/output. The use of control traits guaranties the construction of individuals that are capable and useful to control the robot moves. To diminish the search durations, besides the usual genetic operators like mutation, permutation and crossover, a new atrophy operator was introduced.

Keywords: linear robot, control synthesis, genetic programming, Petri nets, formal language, control traits.

1 Introduction

Robot control generally concerns path planning, decision making and motion control, depending on the problems the applications solve. Robots are enhanced with sensors to get the environment structure or to observe its behavior. Robots have tasks to fulfill and they have to react to environment or other system participant behaviors. Some applications use robots in a Flexible Manufacturing System (FMS) that is capable to process parts performing different activities according to their specified technologies. A FMS can process concurrently different parts involving different sequences of activities. Some types of parts could be manufactured once, and others repeatedly. Flexibility means to change as quickly as possible the scheduling and accept the processing of new parts involving other technologies; meantime the previous demands are not terminated. In such applications, the robots react to the environment demands and changes.

The structure of the considered FMS composed of one single axe linear robot (R) that moves parts to a set of machine tools (positioned in the places p_1, p_2, p_3 and p_4) performing different activities is represented in Fig. 1. The number of places is reduced for presentation purposes. The job of the robot is to precisely

J.-H. Kim et al. (eds.), *Robot Intelligence Technology and Applications 2*,
Advances in Intelligent Systems and Computing 274,
DOI: 10.1007/978-3-319-05582-4_52, © Springer International Publishing Switzerland 2014

set a part in the processing place and then the control system can start the processing activity. Until an activity is finished, the robot can move other parts to different places. The robot moves from one place to another could have different durations depending on the distance it has to cover. The positioning of the robot in front of a place or the demand for a new task can be signaled to the robot control system. The robot obeys to *left*, *right* and *halt* control signals.

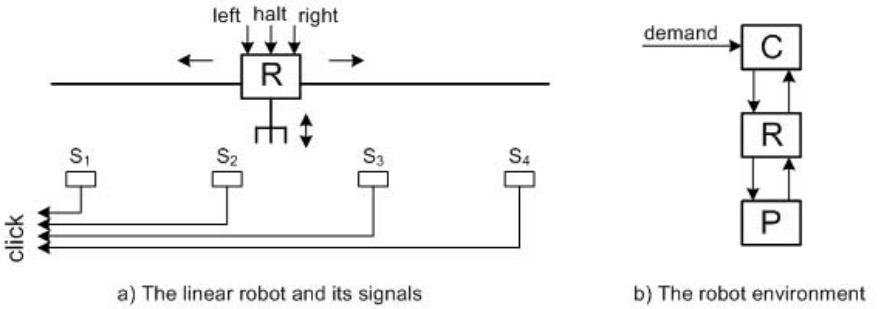

Fig. 1. The application architecture

The controller (C) receives the demands to move parts from one place to another and controls the robot moves taking into account the position signaled by the plant (P).

The current research problem is to synthesize automatically the controller behavior using the robot and plant model and some performance evaluations.

2 Related Works

The current application consisting of a single axe linear robot and some machine tools is transformed into a Discrete Event System (DES). An automatic synthesis control method for a kind of DES is based on bipartite directed graphs that yield the feasible control trajectories and their corresponding states [1]. This is combined with the supervisory synthesis.

The design of logic controllers for event-driven systems that relies on intuitive methods leads to control codes that require extensive verification and are hard to maintain and modify, and may even fail at times. Supervisory control theory provides a formal approach to logic control synthesis. This is used to derive a supervisor that enforces the specifications offering maximum flexibility [2].

2.1 Petri Nets

Some methods of synthesis use Petri nets to prevent the entrance into forbidden state and construct maximally permissive controllers [3].

More complex Petri nets models enhanced with time are introduced like Delay Time Petri Nets (DTPNs) [4] or Time Petri Nets (TPNs) [5]. Reduction methods are used to get more simple models and analysis methods based on reachability serve to model behavior verification.

In the current research relative to the TPN the following semantics are used. The TPN controller models are deterministic and fulfill the following assumptions:

1. The TPN has no conflicts or free elections (see Figure 2).
2. If more than one timed transition is executable from the same marking, the transition with the shortest delay is first chosen for execution;
3. If the TPN model has conflicts or free elections, the following semantics are accepted: the order of transitions chosen for execution is given by their index;
4. The system works with reserved tokens. A transition that started the execution cannot be cancelled by another one with a shorter delay;
5. The places of the input and output interface sets (P_C andP_R in Figure 7) are used exclusively only for input or output operations respectively.
6. The transitions correspond to actions and their executions have no durations.

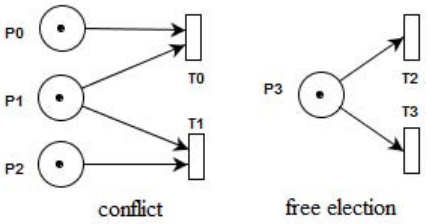

Fig. 2. Conflict and free election TPN structures

2.2 Genetic Programming

The problem of automatic control of a robot model in an arbitrary two-dimensional environment can be obtained based on behavior evolution with the use of GP [6]. This allows different behaviors suited to the environment and user requirements. One of the difficult problem of the GP is the uncontrolled growth of the program size (i. e. bloat). This is usually directly linked to the genotype dimension. Many methods to control the bloat are proposed [7]. An efficient bloat control mechanism is based on examining each function node in the programs. The nodes without contribution are removed before the creation of offspring [8].

3 Robot and Plant Model

The control synthesis requires (is based on) the robot and the plant model. This was constructed taking into account their specifications. The DTPN model is

presented in Figure 3. The delays are not represented to diminish the figure complexity for the presentation and understanding purposes. There can be seen the control signals *left*, *halt* or *right* and the sensor signals activated when the robot reaches a place p_1, p_2,p_3 or p_4. These places model and store the robot position; meantime the upper part of Figure 3 describes the robot behavior when it receives the control signals.

Two plant constructions can be used. One construction signals the reach of each place separately, and another signals when any place is reached. From another point of view, the plant states can be accessible or not. The two constructions involve different efforts for the control synthesis.

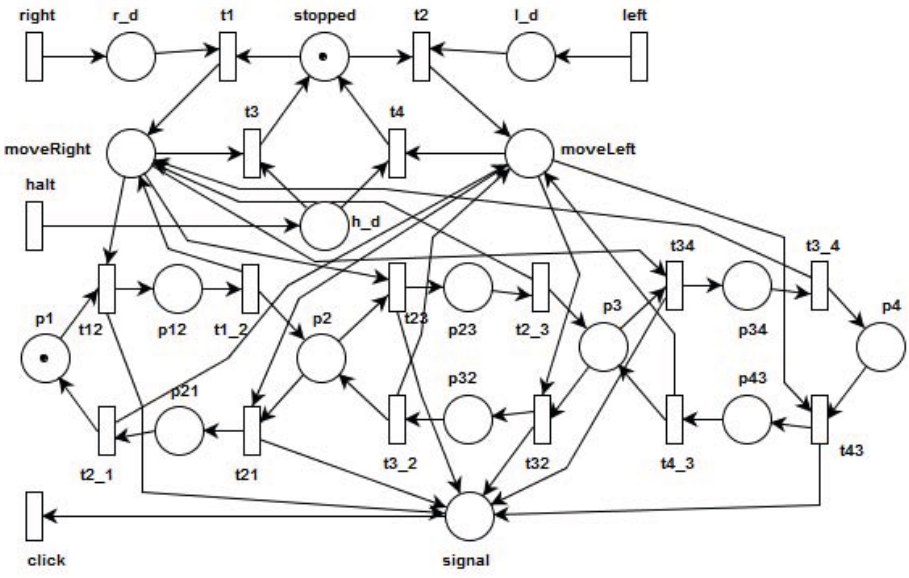

Fig. 3. The robot and the plant DTPN model

4 Control Synthesis

To get the controller behavior the GP was used [9]. The controller behavior is modeled by a TPN. So, the controller solution is a GP individual. As has been mentioned above, GP methods code the individual genotype using a tree structure. The mapping between TPN model and its GP genome is performed by TPNL.

GP operators, selected based on performance evaluation functions, act on genotypes. The performance evaluations use the robot-plant model, the controller TPN model and some parameters that increase the search speed.

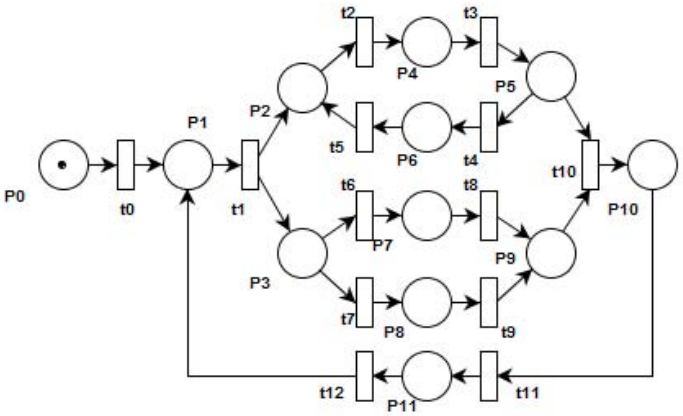

Fig. 4. Example of a TPN

4.1 Time Petri Net Based Language (TPNL)

The role of TPNL is to transform TPN models into string expressions that can be transformed later into the Lisp-S expressions required by the GP algorithms. The transformation of the TPNL expressions into Lisp-S expressions can be performed automatically by changing the operator positions. TPNL uses the following operator symbols:

- '*' for sequence,
- '+' for selection,
- '&' for concurrency and
- '#' for the loop composition.

Considering the example in Figure 4, the following relations describe:

- $t_2 * t_3$ the sequence of t_2 and t_3,
- $(t_2 * t_3)\ \#\ (t_4 * t_5)$ the loop of two sequences,
- $(t_6 * t_8) + (t_7 * t_9)$ the selection between two sequences and
- $((t_2 * t_3)\#(t_4 * t_5))\&((t_6 * t_8) + (t_7 * t_9))$ the concurrent composition.

The sequence generated by the entire TPN can be described by the expression:

$$\sigma = t_0 * ((((t_2 * t_3)\#(t_4 * t_5))\&((t_6 * t_8) + (t_7 * t_9))))\#(t_{11} * t_{12})) \qquad (1)$$

The timings (delays) are not represented in figure and not given in the previous formula for simplification reasons.

TPNL can describe the relation between a controller and a plant. For the model presented in Figure 5 containing a plant and its controller, the descriptions using TPNL are:

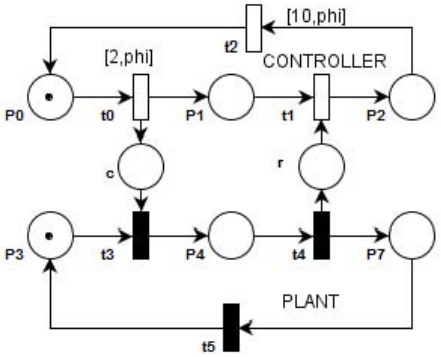

Fig. 5. Controller - plant example

$$\sigma_{controller} = (t_0[2, c] * t_1[r, \phi])\#t_2[10, \phi] \tag{2}$$

$$\sigma_{plant} = (t_3[c, \phi] * t_4[\phi, r])\#t_5[\phi, \phi] \tag{3}$$

The first argument of the transition symbol $t_i[a, b]$ represents a (time) delay or an event signaled by an input channel a. The second argument corresponds to an event signaled by the current component through the output channel b. The ϕ symbol is used to specify an inexistent channel (i. e. the lack of an input event, time or output event).

GP is used here to guide the search of the controller solution through a huge space. TPNL can be used to perform the bijective mapping between genotypes and individuals. For example, for the TPN controller model presented in Figure 5, the TPNL expression can be transformed into the following Lisp-S expression:

$$\sigma_{controller} = \#(*t_0[2, c], t_1[r, \phi]), t_2[10, \phi] \tag{4}$$

The tree representation of the above genotype is given in Figure 6. The TPNL operators become nodes; meantime the transition arguments become leaves.

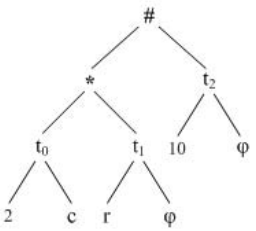

Fig. 6. The simple controller genotype tree representation

4.2 Controller Model Synthesis

Figure 7 depicts the general interfaces between the (robot and) plant and its controller. In the general case of a discrete event system, the plant is modeled by a DTPN and the controller by a TPN. The notations in the figure are:

- P_C denotes the set of control places
- T_C denotes the set of controlled transitions
- T_R denotes the set of reaction transitions and
- P_R the set of reactive places.

For the current problem they are:

- T_C = {right, left, halt}
- T_R = {click}

Another application variant includes the robot position signals, too.

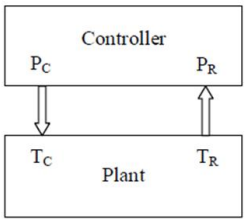

Fig. 7. Controller - plant (the robot is included too) interfaces

The synthesis problem refers to the conceiving of a TPN that has the interface sets P_C and P_R and that maximizes some given performance criteria for the given plant model.

The analysis of the robot-plant model presented in Figure 3 shows that it is a system that has memory. Many discrete event plants provide memory traits. The construction of the plant-controller interfaces can be done to provide the plant current state or not. The current research focuses on the following types of controllers.

First type controller (FTC) uses an interface that provides the plant internal state. This leads to a controller that receives a demand, reacts accordingly and does not (need to) store the plant final state to be able to perform the next request.

Second type controller (STC) does not use an interface that provides the plant internal state, but it is enhanced with a structure that models the plant state. STC is started simultaneously with the plant having specified the plant initial value. STC receives a demand and at the end of fulfilling the requirement, the controller has to store the plant observed state for use in the next demands.

Third type controller (TTC) does not have any information about the plant structure and its initial state value. TTC has to be constructed such that it maintains the plant state information and gets the plant initial state. TTC receives demands, performs their requirements and stores the plant state for future use.

The GP algorithm involves the following activities:

1. the random creation of the initial population
2. the individual evaluations
3. the random individual selection for reproduction
4. the creation of the offspring using randomly genetic operators (mutation, crossover and permutation)
5. the selection of the solution (the individual that won the competition)

Steps 2-4 are executed until a stopping criterion is fulfilled.

The three GP operators act on the genotype trees. The crossover uses two selected individuals (parents), it chooses in each parent a subtree and interchanges them.

Figure 8 shows the TPN models of two selected parents. The TPNL descriptions are:

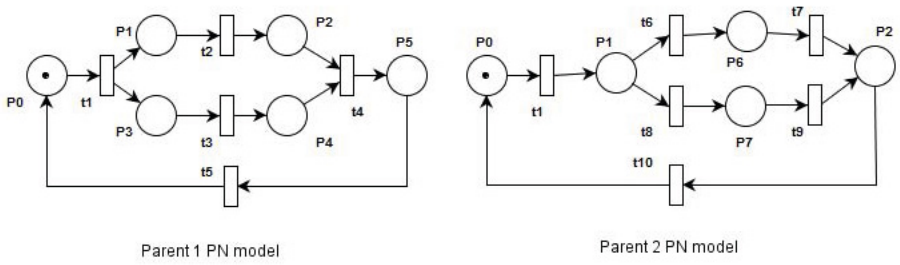

Fig. 8. Two parents PN models

$$\sigma_1 = (t_2 \& t_3) \# t_5 \tag{5}$$

$$\sigma_2 = (t_1 * ((t_6 * t_7) + (t_8 * t_9))) \# t_{10} \tag{6}$$

Transforming the TPNL descriptions into Lisp expressions gives:

$$\sigma_{1Lisp} = \#(\&t_2, t_3), t_5 \tag{7}$$

$$\sigma_{2Lisp} = \#(*t_1, (+(*t_6, t_7), (*t_8, t_9))), t_{10} \tag{8}$$

It is supposed the random crossover chooses the subtree t_2 from Parent 1 and the subtree $(+(*t_6, t_7), (*t_8, t_9))$ from Parent 2 for spring construction. Figure 9 represents the two parent trees and the selected subtrees for crossover.

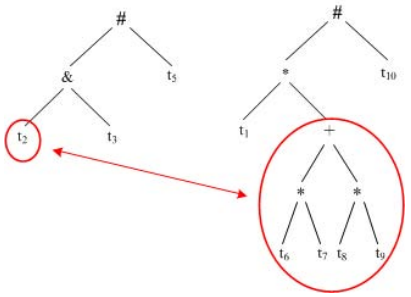

Fig. 9. Parent trees and crossover marking

Performing the crossover, the children described by σ_3 and σ_4 are obtained:

$$\sigma_{3Lisp} = \#(\&(+(*t_6, t_7), (*t_8, t_9)), t_3), t_5 \tag{9}$$

$$\sigma_{4Lisp} = \#(*t_1, t_2), t_{10} \tag{10}$$

$$\sigma_3 = (((t_6 * t_7) + (t_8 * t_9))\&t_3)\#t_5 \tag{11}$$

$$\sigma_4 = (t_1 * t_2)\#t_{10} \tag{12}$$

The children trees are given in Figure 10 and their TPN models in Figure 11.

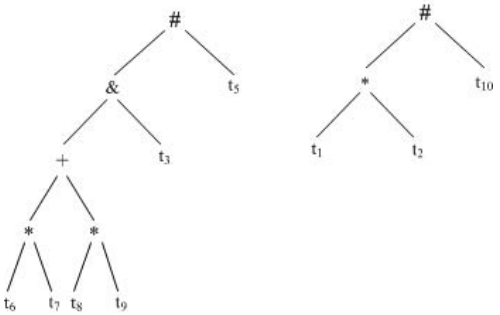

Fig. 10. Crossover resulted children trees

The mutation genetic operator acts on a node changing the TPNL operator, or on a leaf changing the input channel, the delay duration or the output channel. The TPN model presented in Figure 12 is chosen to show the operator mutation.

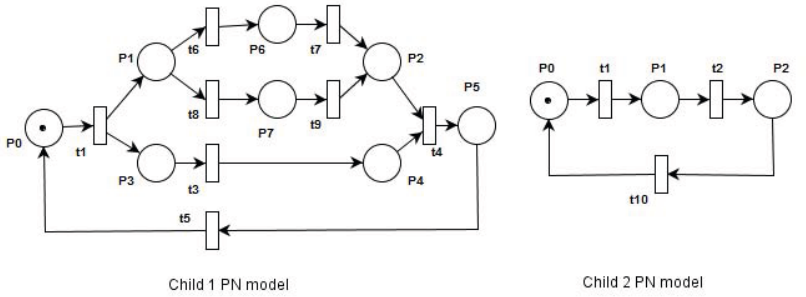

Fig. 11. Children PN models

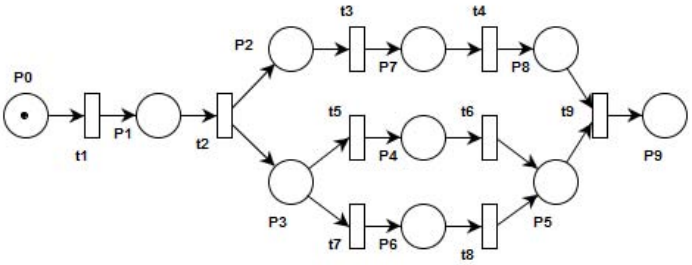

Fig. 12. PN model for operator mutation

The TPNL description of TPN model chosen for operator mutation is:

$$\sigma = t_1 * ((t_3 * t_4)\&((t_5 * t_6) + (t_7 * t_8))) \tag{13}$$

The transformation into Lisp expression provides:

$$\sigma_{Lisp} = *(*t_1, (\&(*t_3, t_4), (+(*t_5, t_6), (*t_7, t_8)))) \tag{14}$$

The corresponding tree is drawn in Figure 13. The node '&' was chosen randomly to be replaced with the operator #.

The Lisp expression of the mutated individual model is:

$$\sigma_{newLisp} = *(*t_1, (\#(*t_3, t_4), (+(*t_5, t_6), (*t_7, t_8)))) \tag{15}$$

The mutated individual model can be transformed into:

$$\sigma_{new} = t_1 * ((t_3 * t_4)\#((t_5 * t_6) + (t_7 * t_8))) \tag{16}$$

Figure 14 presents the TPN model obtained after operator mutation.

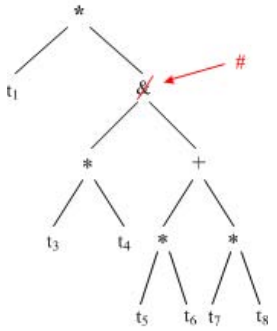

Fig. 13. The tree representation for operator mutation

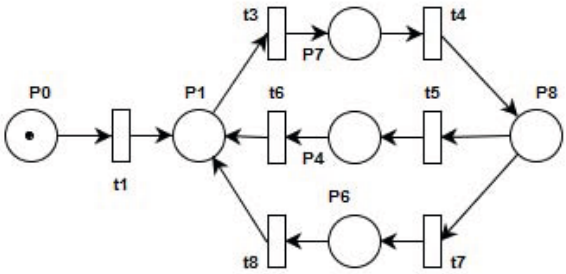

Fig. 14. PN model of operator mutation result

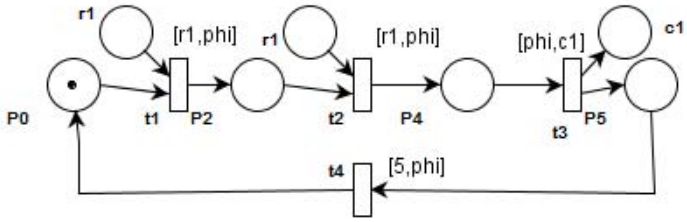

Fig. 15. PN model of leaf mutation

Figure 15 presents a TPN model that has to suffer mutations on transition arguments. The TPNL description and Lisp expression are:

$$\sigma = (t_1[r_1, \phi] * t_2[r_1, \phi] * t_3[\phi, c_1])\#t_4[5, \phi] \tag{17}$$

$$\sigma_{Lisp} = \#(*(*t_1[r_1, \phi], t_2[r_1, \phi]), t_3[\phi, c_1]), t_4[5, \phi] \tag{18}$$

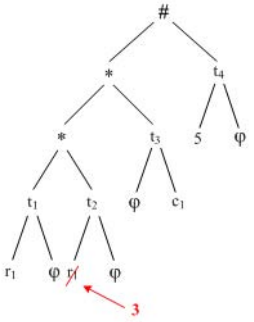

Fig. 16. Tree representation of argument mutation model

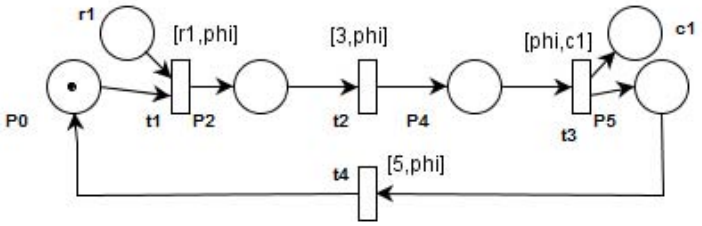

Fig. 17. PN model of argument mutation result

The mutation of the first argument of transition t_2 is supposed to change the input channel r_1 into a 3 time units delay. That is described by:

$$\sigma_{newLisp} = \#(*(*t_1[r_1, \phi], t_2[3, \phi]), t_3[\phi, c_1]), t_4[5, \phi] \tag{19}$$

$$\sigma_{new} = (t_1[r_1, \phi] * t_2[3, \phi] * t_3[\phi, c_1])\#t_4[5, \phi] \tag{20}$$

The mutation is represented on the tree in Figure 16. The result obtained after first argument mutation of the transition t_2 is drawn in Figure 17. Similar results are obtained if the mutation genetic operator acts on the second arguments, but in this case the control signal channel is changed.

The permutation changes within the same individual one node to another node or one leaf to another leaf. The TPN model used to apply the permutation is the Child 1 in Figure 11. It is supposed that the permutation interchanges the operators '&' and '+' that leads to the σ_5 expressions and consequently the tree presented in Figure 18.

$$\sigma_{5Lisp} = \#(+(\&(*t_6, t_7), (*t_8, t_9)), t_3), t_5 \tag{21}$$

$$\sigma_5 = ((t_6 * t_7)\&(t_8 * t_9)) + t_3)\#t_5 \tag{22}$$

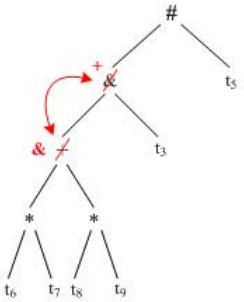

Fig. 18. Permutation representation on the tree

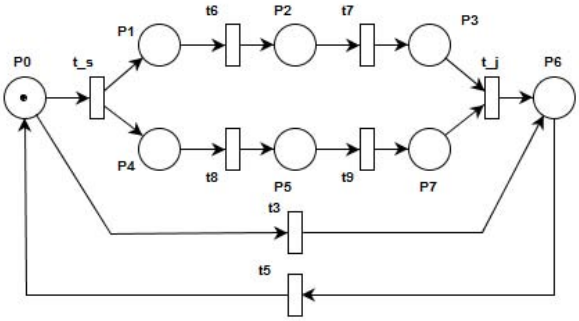

Fig. 19. Resulted PN model after permutation

After the permutation, the TPN model in Figure 19 is obtained.

The pure random creation of an individual could lead to many individuals that will not survive (aborted being useless) due to the fact that they do not have at least the compulsory control traits. For this reason some control traits are conceived and they have specified probabilities to be used for initial individual creation or by mutation operator during the reproduction phase. These are introduced in the so called trait pool.

In the current research the used trait pool is presented in Table 1. There t_i represents transitions and $\sigma_i (i = 1, 2)$ correspond to sequences of transitions. Different probabilities are assigned for trait use depending on the used model of the plant.

4.3 Controller Behavior Evaluation

According to GP the fitness function plays an important role for the individual selection and this guides indirectly the search for the desired solution. The evaluation is achieved using the concurrent simulation of the (robot plus) plant

Table 1. Trait pool

Trait	Coding	Probability
input	$t_i[a, \phi]$	12
output	$t_i[\phi, b]$	12
input-output	$t_i[a, b]$	12
loop	$\sigma_1 \# \sigma_2$	10
concurrency	$\sigma_1 \& \sigma_2$	10
selection	$\sigma_1 + \sigma_2$	10
memory	$(t_i[r, s] * t_j[\phi, \phi]) \# (t_l[l, s] * t_k[\phi, \phi])$	10
read (state 0)	$t_1[0, c_3] * t_2[r_0, \phi]$	5

and controller models. In the current case, the controller receives a specified test sequence (demands) containing the demanded movements.

An example of a demanded sequence used for controller evaluation is:

$$demand = p_3[0, \phi] * p_1[5, \phi] * p_2[4, \phi] * p_4[3, \phi] * p_1[6, \phi] * p_3[4, \phi] * p_2[2, \phi] * p_1[3, \phi] * p_4[0, \phi] * p_1[7, \phi] * p_3[4, \phi] * p_2[3, \phi] * p_3[3, \phi] \quad (23)$$

The durations of demands should be chosen such that the robot can perform them under right controller supervision.

Two criteria can be used for individual evaluations:

- individual behavior evaluation
  - robot reaches or not the demanded state in a specified duration
  - number of transitions executed by plant or controller to reach the destinations (i. e. the time to fulfill a demand)
  - robustness relative to plant reaction delays
  - controller reaction delays
- individual structure evaluation
  - number of transitions
  - number of channels used for the interfaces plant-controller or plant-supervisor
  - number of parallel threads of execution for controller implementation
  - number of loops etc.

In the current research they are expressed using the following formulas:

$$Criterion_1 = \alpha_1 \cdot targetReached - \alpha_2 \cdot transitionExecuted - \alpha_3 \cdot duration \quad (24)$$

where:

- targetReached means that when the controller signals the end event, the plant reached the target,
- transitionExecuted counts how many transitions were executed to reach the target,
- duration counts the time (clock tic) from the start until the end signal of each demand

$$Criterion_2 = -\alpha_4 \cdot transitionNo - \alpha_5 \cdot channelNo - \alpha_6 \cdot threadNo - \alpha_7 \cdot loopNo \quad (25)$$

where:

- transitionNo represents the number of transitions used for individual construction,
- channelNo corresponds to the number of channels used by the controller,
- threadNo counts the number of threads and
- loopNo counts the number of loops.
- $\alpha_i, i = 1, \ldots, 7$ are weight coefficients.

4.4 Increasing the Search Speed

The search guiding can be achieved using mono or multi-objective functions. These are used for the individual selection. On the other hand, the probabilities of individual selection for applying the genetic operators also influence the search directions and speeds. Different probabilities can be assigned to perform the individual initial construction or modifications during the evolution.

In the case of GP, it can be assigned probabilities for operator selections and constraints for the number of the operators of specified types that are accepted in any individual during the construction of the initial population or the mutation operation.

The two criteria proposed to guide the solution search are used for individual selection, but the constrained numbers of the GP operators are involved too.

Besides the usual GP operators a new operator called *atrophy* was introduced due to the fact that during the individual evaluations some transitions are not used. The atrophy operator is applied (unlike the usual GP operators) after the individual selection and it has the role to remove from the individual the transitions that are not involved in any behavior demand. This removal further affects the individual genotype.

To avoid the unlimited increase of an individual (that leads to bloat) and the excessive use of some TPNL operators besides the choosing probabilities, some constraints have been assigned as can be seen in Table 2. On the other hand, GP operators can have different selection probabilities depending on the criterion used for individual selection as presented in Table 3. The tests were performed for a population of 2000 individuals and 300 generations. Table 4 contains the parameter values of the evaluation criteria. Their value can increase or decrease the search speed too.

5 Tests and Results

The controller synthesis involves the construction of the environment and the individuals, followed by the phenotype evaluations.

The environment was obtained by the implementation of the robot DTPN model. This requires two matrices with dimensions equal to the number of places

Table 2. Operator probabilities and constraints

Operator	Probablity	Constraint
*	40	50
+	20	20
&	5	50
#	10	10

Table 3. Genetic operator numbers used for the different criteria on a population of 2000 individuals

Criterion	Mutation	Crossover	Permutation
$Criterion_1$	285	850	150
$Criterion_2$	190	450	75

and the number of transitions. Two delay vectors were used, one with the dimensions equal to the number of transitions and the other equal to the number of places.

The individual constructions were achieved by the implementation of the TPN models that require two matrices with dimensions equal to the number of places and the number of transitions (like for the DTPN model) and a delay vector with the dimension equal to the number of transitions.

The phenotype evaluation was performed in simultaneous simulation of the environment, individuals and their interactions.

Two separate tests were performed on a personal computer with a 2.6GHz dual core processor:

1. The considered plant has an additional structure that can output the exact state of the plant (i. e. the robot position). The read command is denoted c3. The generated controller has 49 transitions and is made up of traits and individual transitions, connected with operators. The performance function evaluates the behavior of the (plant + controller) system in 16 different scenarios (the plant is in state i, the command is to go to state j, i,j=1..4). The solution is generated in 300 generations and it took around 9.5 hours to run. The resulted Lisp-S expression of the solution is (+(+(+(*(*(t1,r5,c3)(t2,0,c1))(+(+(+(t3,r0,c3)(*(t4,1,c3)(t5,r2,fi)))(t6,r1,fi))(+(t7,1,c3)(t8,r0,fi))))(*(*(t9,r6,c3)(t10,1,c3))(+(+(*(*(*(*(t11,r0,fi)(t12,0,c0))(*(t13,1,c3)(t14,r1,fi)))(t15,0,c2))(t16,r1,fi))(*(*(*(t17,r2,fi)(t18,0,c1))(*(t19,2,c3)(t20,r1,fi)))(t21,0,c2)))(*(*(*(*

Table 4. The value of the coefficients of the evaluation functions

Coeficient	α_1	α_2	α_3	α_4	α_5	α_6	α_7
$Value$	350	80	10	20	10	10	10

(t22,r3,fi)(t23,0,c1))(*(t24,1,c3)(t25,r2,fi)))(*(t26,1,c3)(t27,r1,fi)))
(t28,0,c2)))))(*(*(t29,r7,c3)(t30,1,c3))(+(+(+(*(*(*(*(t31,r0,fi)
(t32,0,c0))(*(t33,1,c3)(t34,r1,fi)))(*(t35,1,c3)(t36,r2,fi)))(t37,0,c2))
(*(*(*(t38,r1,fi)(t39,0,c0))(*(t40,1,c3)(t41,r2,fi)))(t42,0,c2)))(t43,r1,fi))
(*(*(*(t44,r3,fi)(t45,0,c1))(*(t46,1,c3)(t47,r2,fi)))(t48,0,c2)))))(t49,0,c0))

2. The plant is without the additional structure, the controller is forced to track the state of the plant. Using this strategy, no result was generated in 300 generations. However, if we add the additional structure (memory) to the trait pool (such as to let the GP algorithm include it in the controller), the generated solution contains 39 transitions and is generated in 12 hours running time. Here, P1,..., P4 represent the places where the state of the plant is being stored (these places have been given as input channels for the rest of the controller). The best solution is
(+(+(+(*(t1,cmd1,fi)(+(+(+(t2,P1,stop)(*(t3,P2,left)(t4,P1,stop)))
(*(*(t5,P3,left)(t6,P2,fi))(t7,P1,stop)))(*(*(*(t8,P4,left)(t9,P3,fi))(t10,P2,fi))
(t11,P1,stop))))(*(t12,cmd2,fi)(+(+(+(*(t13,P1,right)(t14,P2,stop))
(t15,P2,stop))(*(t16,P3,left)(t17,P2,stop)))(*(*(t18,P4,left)(t19,P3,fi))
(t20,P2,stop)))))(*(t21,cmd3,fi)(+(+(+(*(*(t22,P1,right)(t23,P2,fi))
(t24,P3,stop))(*(t25,P2,right)(t26,P3,stop)))(t27,P3,stop))(*(t28,P4,left)
(t,P3,stop)))))(*(t29,cmd4,fi)(+(+(+(*(*(*(t30,P1,right)(t31,P2,fi))(t32,P3,fi))
(t33,P4,stop))(*(*(t34,P2,right)(t35,P3,fi))(t36,P4,stop)))(*(t37,P3,right)
(t38,P4,stop)))(t39,P4,stop))))

6 Conclusions

The newly introduced control traits diminish significantly the solution search duration. A control designer usually knows the main traits of the expected controller. But the probabilities of the trait appearances are unknown. These are problem dependent.

The proposed method leads to a TPN that is equivalent to a set of interconnected state machines which can be easily programmed. The obtained controller can be implemented using a single or multi threading execution.

The TPN solution can be easily implemented on a FPGA (Field Programmable Gate Array) or on a micro-controller using their programmable languages. The execution durations of the corresponding programs are very short relative to the robot temporal behavior, so it can be stated that a real-time controller has been obtained.

The proposed synthesis method can be successfully applied to any discrete event system. The controller designer should focus its efforts on specifying the controlled part of the application and the control performance evaluation, instead of finding a control algorithm that fulfills the specification.

References

1. Kapkovic, F.: Automatic control synthesis for agents and their cooperation in MAS. Computing and Informatics 29, 1045–1071 (2010)

2. Chandra, V., Zhongdong, H., Kumar, R.: Automated control synthesis for an assembly line using discrete event system control theory. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 33(2), 284–289 (2003)
3. Dideban, A., Alla, R.: Controller synthesis by Petri nets modeling. In: Proc. of the 3rd International Workshop on Verification and Evaluation of Computers and Communication Systems (2010)
4. Juan, E.Y.T., Tsai, J.P., Murata, T., Zhou, Y.: Reduction methods for real-time systems using delay time Petri nets. IEEE Transactions on Software and Engineering 27(5), 422–448 (2001)
5. Wang, J., Deng, Y., Xu, G.: Reachability Analysis of Real-Time Systems Using Time Petri Nets. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 30(5), 725–736 (2000)
6. Paic-Antunovic, L., Jakobovic, D.: Evolution of automatic robot control with genetic programming. In: Proceedings of the 35th International Convention MIPRO, pp. 817–822 (2012) ISBN: 978-1-4673-2577-6
7. Alfaro-Cid, E., Merelo, J.J., Fernndez de Vega, F., Esparcia-Alczar, A.I., Sharman, K.: Bloat Control Operators and Diversity in Genetic Programming: A Comparative Study. Evolutionary Computing 18(2), 305–320 (2010)
8. Song, A., Chen, D., Zhang, M.: Bloat control in genetic programming by evaluating contribution nodes. In: GECCO 2009: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1893–1894 (2009)
9. Letia, T.S., Hulea, M., Cuibus, O.: Controller synthesis method for discrete event systems. In: IEEE International Conference on Automation Quality and Testing Robotics (AQTR), pp. 85–90 (2012), doi:10.1109/AQTR.2012.6237680