Developing a Petri Nets based Real-Time Control Simulator

Reggie Davidrajuh
Electrical and Computer Engineering
University of Stavanger
Stavanger, Norway
e-mail: reggie.davidrajuh@uis.no

Abstract — **Petri net is a powerful tool for modeling and simulation of discrete event dynamic systems (DEDS). This paper is about developing a new real-time control simulator based on Petri net. This paper presents some of the design challenges that were dealt with during the development stages of the simulator; the design issues were: realization of token gaming (time requirements in consumption of input tokens and creation of output tokens), representation of event continuity (whether places or transitions represent event continuity), and interaction with the external influences (whether to map places or transitions to interact with external events). This paper shows that the following were found to be the most suitable for development of the real-time control simulator: ASAP firing for realization of token gaming, transition for representing continuity of events, and transition (with the pre and post definitions) for interacting with external event).**

Keywords - modeling and simulation, discrete event dynamic system (DEDS), Petri net simulator, real-time systems, GPenSIM

I. Introduction

General Purpose Petri Net Simulator (GPenSIM) is a tool for modeling and simulation of discrete-event dynamic systems (DEDS) [1]; GPenSIM is based on Petri nets [2; 3]. GPenSIM has been successfully used as a tool for teaching discrete simulation and then also to solve some industrial problems [4; 5].

This paper talks about the recent real-time enhancements added to GPenSIM to make it function as a Real-Time Control Simulator. This paper talks about some of the design issues that were dealt with during the development of the real-time version. In the following section, a short introduction to existing version of GPenSIM is given. From section III, real-time enhancements to GPenSIM are discussed.

II. GPenSIM

This section presents a brief introduction to GPenSIM.

A. GPenSIM: an Easy-to-use Tool for DEDS Simulation

The reasons for developing a new simulator were [6; 7]:

For basic users: 1) a tool that is easy to understand and easy to use, even for users with minimal mathematical and programming skills.

For advanced users: 2) allow seamless integration of DEDS models with the other toolboxes that are readily available on the MATLAB platform; 3) allow easy extension of GPenSIM functions, and 4) Compact simulation code.

GPenSIM is developed by the author of this paper, in order to satisfy the four criteria stated above (ease of use, flexible, extensible, and compact code). GPenSIM is realized as a toolbox for the MATLAB platform, so that diverse toolboxes that available in the MATLAB environment (e.g. Fuzzy Logic Toolbox, Control Systems Toolbox, Advanced Statistical Analysis Toolbox, etc. [8]) can be used in the discrete models that are developed with GPenSIM.

B. Existing Tools for Real-Time Control Simulation

There are many tools that are available for real-time control simulation. SimEvents and SIMULINK are excellent tools that run on the MATLAB platform. Both are capable of real-time control and allow incorporation of diverse tools (fuzzy, neural net, etc.) and libraries; however, these two tools are general purpose and are not Petri net based. The project this paper talks about is only interested in developing a Petri net based simulator because of the many benefits of using Petri nets (such as possessing simple graphical representation as well as strong linear algebraic representation that allows mathematical analysis both on the structural and on the behavioral front) for modeling and simulation of DEDS. LabView is another good tool that can be defined as a real-time control simulator; however, LabView is not based on Petri nets either. In summary, to the author's best knowledge, there isn't a single tool that is Petri net based, real-time capable and allows easy integration of diverse tools and techniques in the models of DEDS.

C. The Existing Architecture of GPenSIM

A simple model created GPenSIM has 3 files: the elements of the Petri net model (places and transitions) and the static connections (arcs) that exist between the elements are defined in a file called Petri net Definition File (PDF); initial inputs (initial tokens, etc.) to the system are defined in the main simulation file (MSF), and the interaction between the Petri net model and the environment are defined in file(s) called Transition Definition File (TDF). Interested reader is referred to the GPenSIM user manual [1].

Figure 1 shows the architecture of GPenSIM; models that are developed with GPenSIM consist of a number of files (M-files or MATLAB files) [1]:

Petri Net Definition Files (PDF): these files define the static structure (elements and their connection) of the discrete model.

Main Simulation File (MSF): this is the file that calls the other files; in this file, initial conditions and variables are declared, and then the simulation is run; finally, the simulation results (arcs) are plotted and analyzed.

Transition Definition Files (TDF): TDF files are to code the conditions ('guard conditions') that must be satisfied before an enabled transition can fire. However, the TDFs are also the files where one can implement a fuzzy logic based inference engine or an inference mechanism that make decisions based on statistical analysis of massive data. There are two types of TDFs:

Individual (for a specific transition) PRE and POST files: a PRE TDF file checks whether guard conditions are satisfied before an enabled transition can fire; it can also reserve resource for use by the transition; a POST TDF file can release resources used by the fired transition and can do book keeping work (statistics) after the transition firing.

COMMON PRE and POST files: these files are common for all the transitions; a COMMON PRE can, for example, reserve a resource that is generally used by any firing transition; likewise, a COMMON POST can release the commonly used resource after firing of a transition.

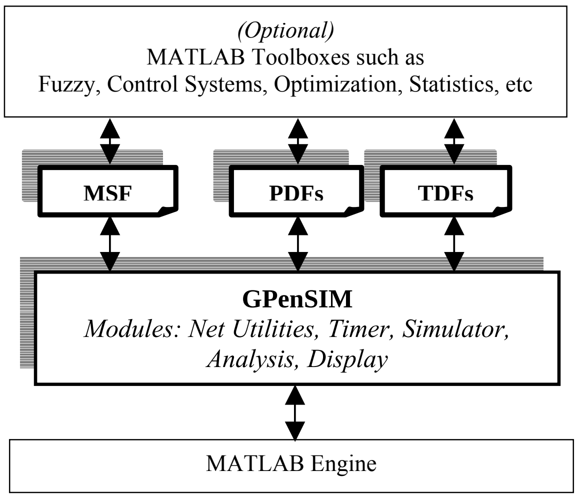
(Optional)
MATLAB Toolboxes such as
Fuzzy, Control Systems, Optimization, Statistics, etc
MSF
PDFs
TDFs
GPenSIM
Modules: Net Utilities, Timer, Simulator, Analysis, Display
MATLAB Engine

Fig.1. Files for simulation

III. DEFINING REAL-TIME GPENSIM

A new definition is given below for modeling, simulation, and control of real-time DEDS using Petri Nets.

Definition: A Real-Time Petri Net consists of four parts:

RTPN = (PN, SE, CE, Time)

The four parts are explained in the following subsections.

A. PN: the Graph Theoretic Structure

PN is the first part of RTPN which defines the graph theoretic structure – the *ordinary P/T Petri Net*, consisting of a set of passive elements called places (P), a set of active elements called transitions (T), a set of bipartite arcs with arc weights connecting places and transitions, and a set of initial markings (m_0).

B. SE: the Extension for System Resources

SE is the second part of RTPN which is the extension for System Resources; System resources are managed in background by the system; *SE* consists of the following elements:

$$SE = (R, \Omega, o, t, X, Y),$$

R is the set of systems resources, $R = \{R_1, R_2, \ldots , R_n\}$

Ω is the set of operations executable by all the systems resources, $\Omega = \{\omega_1, \omega_2, \ldots, \omega_m\}$

o is the partitioning of the operations into subsets that can be executable by the individual resources, $o: R \rightarrow 2^{\Omega}$

t is the set of timing that are taken for operations, $t: R \times \Omega \rightarrow N^+$

X and Y are the set of input and output servers that feed material into the resources and remove worked parts from the resources

C. CE: the Color Extension

The third part of RTPN ***CE*** is the color extension; *CE* has two main functions: overloading tokens with data to differentiate one token from another, and mapping transitions to logic functions:

$$CE = (C, \psi),$$

$C: M \rightarrow \{c_1, c_2, \ldots c_k\}$, *k is the number of tokens.* C is a function mapping the set of tokens (markings) onto a set of value assignments (colors), so that the tokens can carry work-in-progress (W-I-P) information with them.

$\psi: T \rightarrow LOG$ is the set of logical conditions for firing (firing conditions), and for executing resource manipulation commands such as request for resource, resource usage, and release of resource after usage.

D. The Time Factor

The fourth part of RTPN is the timing: ***Time*** represents the timing taken by the various operations, activities, and events in the DEDS. In a Petri net model, timing can be assigned to places, transitions, or to both.

$$Time = (TimeT, TimeP),$$

TimeT: $T \rightarrow \Re$ is a function mapping the set of transitions onto a set of real numbers, where the real numbers represent the firing time – the time taken by transitions to fire. Firing time can deterministic or stochastic.

TimeP: $P \rightarrow \Re$ is a function mapping the set of places onto a set of real numbers, where the real numbers represent the holding time – the time taken by places to hold a token. Holding time can deterministic or stochastic.

IV. Basic Design Issues of Real-Time GPenSIM

This section discusses three main design issues during the development of Real-Time GPenSIM:

1. How to realize 'token game': by ASAP firing, delayed firing, or by aging?
2. How to model events: by places or by transitions? and,
3. How to interact with the external environment: thorough 'loaded places' or by 'firing transitions'?

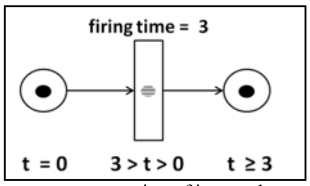

Fig.2. Token game: consumption of input tokens and creation of output tokens

A. Realizing Token Game

As per definition of the ordinary (P/T) Petri nets [2], if a transition has the input places with enough tokens, then the transition is enabled and that it can fire; firing of a transition leads to consumption of the input tokens from the input places. This means, the tokens inside the input places disappears into the firing transition for a while; when the transition completes firing, the output tokens are deposited into the output places, thus the tokens 'reemerges' albeit in different places (in output places) and in different numbers (number of output tokens deposited into the output places depends on the arc weights of the output arcs connecting the output places to the transition). For example, in the figure 2, at time t=0, transition T is enabled (because it has an input place with one token in it) and when it fires, it consumes the input token from the input place; figure 2 explains the disappearance of the input token as it is being consumed (or processed) inside the transition. When the transition T completes firing at time t=3, the transition deposits a token (an output token) into the output place. As the figure captions, the phenomenon of 'disappearance' and 'reemergence' of tokens is known as the 'token game'.

The token game creates some problems, especially in terms of reachability analysis [9-11]: during simulations, most of the states are not reachable due to the firings of the transitions and the ensuing disappearances of tokens; thus, there are generally three methods to realize consumption of tokens during firing of the transitions, the latter two methods avoid token disappearance [9-11]:

ASAP firing of transitions: in this realization, tokens do disappear and remerge as discussed above and shown in figure 2.

Delay before firing: in this realization, when a transition is enabled, the input tokens in the input places are marked as unavailable and will be not be taken away from the input places. After a predefined time delay, the transition fires and thus removes the input tokens from the input places. The delay in transition firing - the time difference between the enabled position and the start of firing - should give ample time for recording the state of the system. This realization requires that all the transitions are assigned delay timing.

Aging of tokens: as in the previous realization, in this realization too when a transition is enabled, the input tokens in the input places are marked as unavailable and will not be taken away from the input places. However, there is no delay in the transition firing as the transition is allowed to fire immediately. In addition, when the transition fires, the tokens in the input places are still there, not removed from the input places, and marked as unavailable. Only when the transition completes firing, the input tokens are removed from the input places and the output tokens are deposited into the output places; thus, creation of output tokens from input tokens happens instantaneously thus all the tokens are accounted for.

GPenSIM realizes the first method for token gaming – ASAP firing of transitions; it was argued that although token gaming creates problems with reachability analysis, ASAP

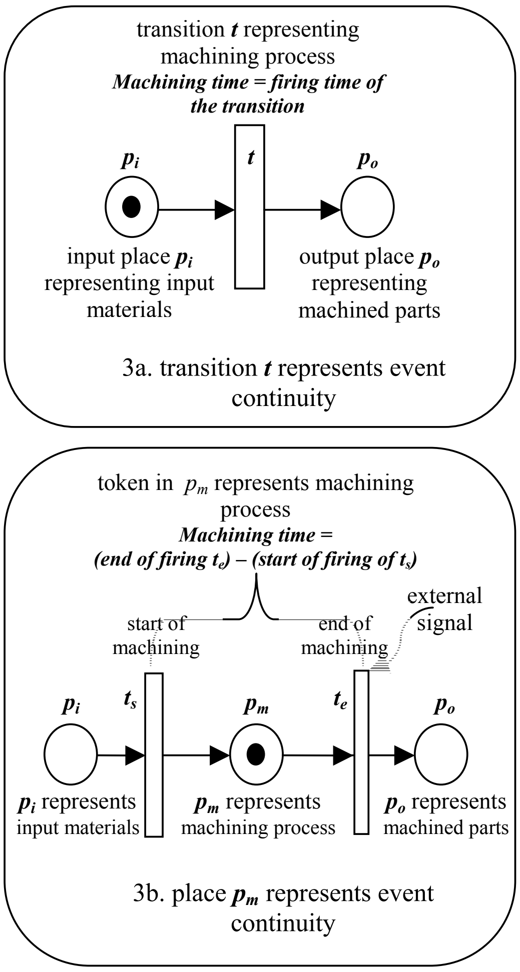
transition *t* representing machining process
Machining time = firing time of the transition
p_i
t
p_o
input place p_i representing input materials
output place p_o representing machined parts
3a. transition *t* represents event continuity
token in p_m represents machining process
Machining time = (end of firing t_e) – (start of firing of t_s)
external signal
start of machining
end of machining
p_i
t_s
p_m
t_e
p_o
p_i represents input materials
p_m represents machining process
p_o represents machined parts
3b. place p_m represents event continuity

Fig.3. Representing events

firing of transitions is the natural way of realizing token gaming: for example, in a production system, when a machine ('transition') is ready ('enabled transition'), input materials ('tokens') from the input conveyor belts ('input places') are consumed by the machine ('firing transition'); the input materials do disappear inside the machine during the machining process. Only when the machining is complete, the machined parts ('output tokens') are deposited into the output conveyor belts ('output places').

B. Representing events: by places or by transitions?

In a Petri net model, an event (or continuity of an event) can be modeled by two ways: an event can be represented by a transition, or by a place.

Figure 3 depicts the two ways of the representation. In figure 3a, a transition represents the machining event: the start of transition firing indicates the start of the machining process and the completion of the firing of the transition indicates the finishing of the machining process. Real-Time GPenSIM generally assumes that events should be represented by transitions as transitions are the active elements in Petri nets. In figure 3b, a place represents the machining process: token(s) inside the place indicates that the machining process goes on.

Though Real-Time GPenSIM generally assumes transitions representing event continuity, it allows both ways of representing continuity of an event. If the machining time is deterministic and known before hand, then the machining is best represented by a transition. If the machining time is not known before and is only determined (or controlled) by external environment, then a place representation (as shown in figure 3b) is better suited for this situation.

C. Interacting with External Environment

A real-time simulator should interact with external environment; it should perceive the external environment through sensors and make it control policies executed through actuators connected with it.

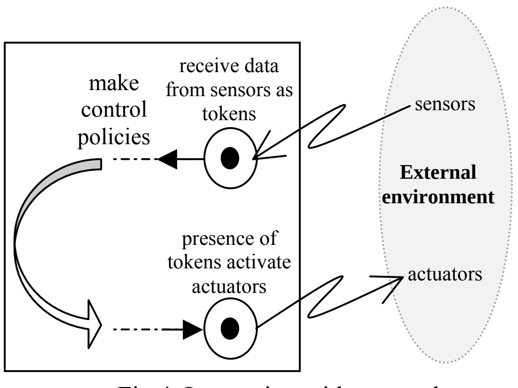
make control policies
receive data from sensors as tokens
sensors
External environment
presence of tokens activate actuators
actuators

Fig.4. Interacting with external environment through *loaded tokens*

There are generally two ways to realize the interaction with the external environment: *loaded places* [12], and *firing transitions*.

Figure 4 shows the way the 'loaded' places are used to interact with the external environment. Sensory data is fed into the places as tokens. Similarly, when a place is loaded with tokens, it triggers actuators executing the control policy coded in the tokens.

The other way is to use transitions. A firing transition may trap sensory data (read data from sensors) and also send output data to actuators.

In GPenSIM, 'firing transitions' are used for interaction with the external environment. Again, this is because of the fact that the transitions are the active elements in a Petri net, thus, are useful as agents for communication with the external environment. To support interaction, an enabled transition can call its preprocessor (TDF PRE) before it starts firing, and after firing it can call a post-processor (TDF POST); a firing transition uses the pre and post processors to interact with the external environment. Figure 5 explains the use of the pre and post processors of a firing transition.

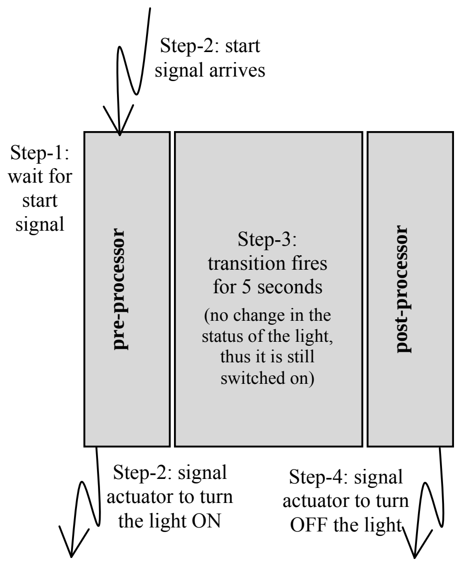
Step-2: start signal arrives
Step-1: wait for start signal
pre-processor
Step-3: transition fires for 5 seconds
(no change in the status of the light, thus it is still switched on)
post-processor
Step-2: signal actuator to turn the light ON
Step-4: signal actuator to turn OFF the light

Fig.5. Using transition to interact with the external environment

Figure 5 depicts a simple example where transition t is supposed to set a light bulb on for a time period of 5 seconds. However, the transition t must wait until it receives a start signal from the sensor. To implement this example:

Step-1: transition t is enabled and waiting for the start signal: the transition is enabled (because it has a token in its input place), thus it runs the pre-processor. In the pre-processor, there is a condition which says that the transition cannot start firing until it receives the start signal from an external sensor. Thus, the arrival of sensory data (start signal) is a logic condition coded in the pre-processor, which must be satisfied for the transition to start firing.

Step-2: arrival of the start signal: when the start signal arrives, the transition t starts firing. Before it starts firing, it runs the final code in the pre-processor, which says to feed the actuator to turn on the light. Hence, the transition sends signal to the actuator to turn on the light.

Step-3: transition t is firing: transition t is active (fires) for a pre-defined and deterministic time period (5 seconds). During this period, there is no change in the status of the light, thus it is still switched on.

Step-4: transition t has fired (transition is completed); now, the transition can run the post processor. Code in the postprocessor informs the transition to force the actuator to turn off the light. Hence, the transition sends signal to the actuator to turn off the light.

V. Further Design Issues of Real-Time GPenSIM

There were two further enhancements made to GPenSIM to make it as a real-time simulator:

1. The usage of "*Real-Time*" clock, and
2. Implementation of *Scheduling* and issues related to scheduling of system resources, such as priorities, reservation policies, algorithms, etc.

A. Real-Time Clock

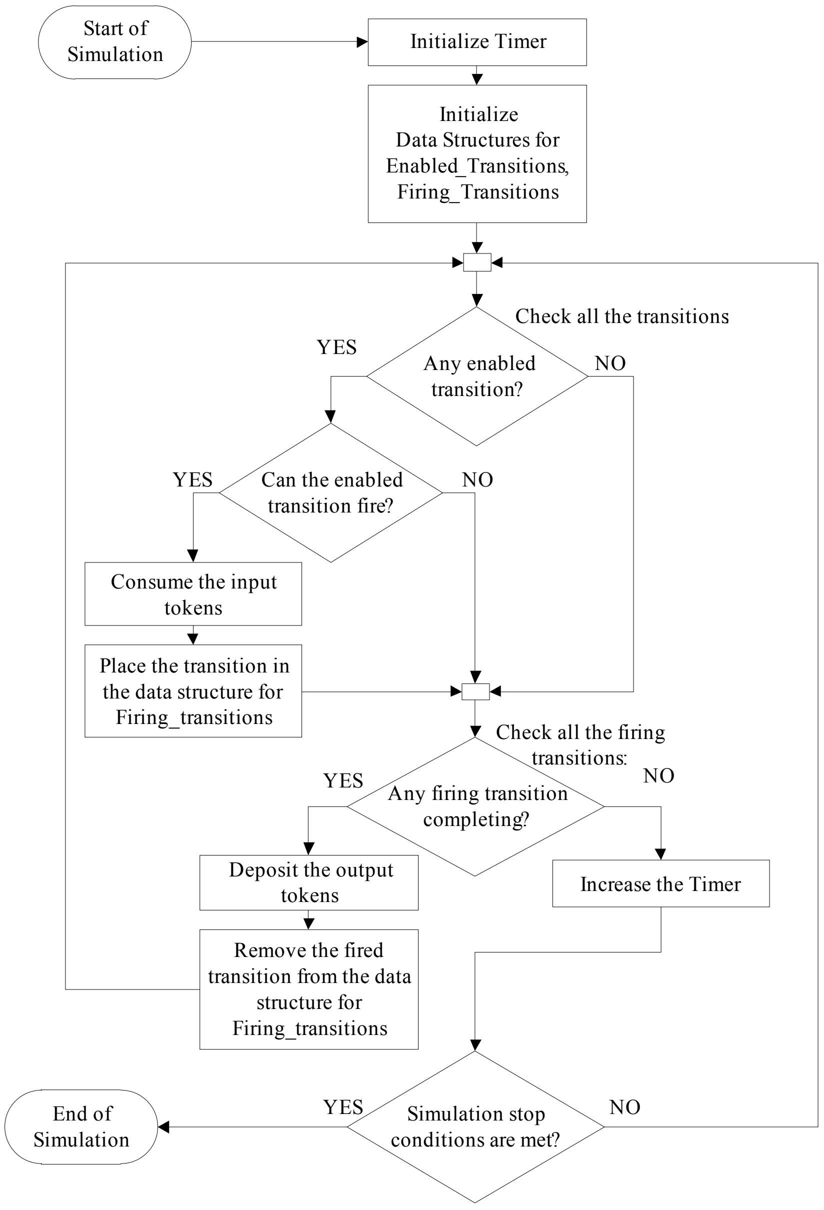
Start of Simulation
Initialize Timer
Initialize Data Structures for Enabled_Transitions, Firing_Transitions
Check all the transitions
YES
Any enabled transition?
NO
YES
Can the enabled transition fire?
NO
Consume the input tokens
Place the transition in the data structure for Firing_transitions
Check all the firing transitions:
YES
Any firing transition completing?
NO
Deposit the output tokens
Increase the Timer
Remove the fired transition from the data structure for Firing_transitions
End of Simulation
YES
Simulation stop conditions are met?
NO

Fig.6. The main simulation loop of GPenSIM

The results of pre-processor turning on the light, the transition keeping the status of light (ON) for 5 seconds, and the post-processor turning off the light – makes the light bulb turned on for 5 seconds.

Figure 6 shows the main loop of the simulation cycles. In the main loop the following steps executed in sequence:

Step-1: The simulator checks for any enabled transition; if there is any enabled transition, it is placed into a queue-like data structure called "*Enabled_Transitions*".

Step-2: If there are any enabled transition (structure Enabled_Transitions is not empty), then the respective TDF of these transitions are run to make sure whether they can actually fire (meaning, the transition satisfy all the conditions defined in the TDF). If an enabled transition can fire, then it is moved from the structure Enabled_Transitions into another data structure called "*Firing_Transitions*". At the same time, input tokens for this transition are consumed from the respective input places.

Step-3: The simulator checks whether any firing transition is completing. This is done by checking the current time against the completion times of the transitions sitting in the structure Firing_Transitions. If there is any transition that is completing, then it is removed from the structure Firing_Transitions and the output tokens are deposited into the output places of the transition. After the firing, there may be some new transitions that are enabled by the arrival of the new output tokens; hence the loop jumps to step-1.

Step-4: To prepare for the next loop, the timer is increased. The timer is increased by a value that is optionally given by the user. Otherwise, the timer increment (known as "*delta_T*") will be one-fourth (0.25 times) of the smallest firing time of any transition. If all the firing times are zero, then the increment will be 0.25 time units. As usual, smaller the increment value for timer, finer the simulations will be (more simulation samples per time unit); on the other hand, larger the increment value for timer, coarser the simulations will be.

After timer increment, the loop jumps to step-1.

Steps 1-4 stated above describes the main simulation cycle. Among the steps 1-4 stated above, step-4 is exclusively deals with timer increment; at the end of each cycle, the timer is incremented by a discrete value (a default value set by the simulator or an optional value chosen by the user). However, in real-time environment, there is no need for sudden increment to timer, as a real-time timer should continuously shift forward. Thus, step-4 becomes obsolete in real-time environment. Instead, whenever the current time is needed, the CPU time (real-time) must be referenced. This change is now implemented in GPenSIM for real-time simulations.

B. *Scheduling of System Resources*

System resources and optimal management of them are very important issues that have to be supported by a real-time system. Thus, real-time GPenSIM also provides the following functionality [7]:

1. Declaring system resources,
2. Utilizing resources (functions for requesting (reserving), allocating, and releasing resources),
3. Declaring Priorities of different transitions,
4. Changing priorities of transitions: functions for increasing or decreasing priority of a transition, and comparing priorities of transitions, and
5. Reporting resource usage: new print functions that show total resource usage, idle time, etc.

The following fundamental assumption was made in realizing the additional functions for system resource modeling: *A system resource is a 'critical section' meaning a resource can be used by only one consumer at a time; this means, resources possesses 'mutual exclusion' property.*

(Though a resource can be used by only one consumer at a time, a consumer can use as many resource as it wants, limited only by availability).

VI. Application Example

This section presents an application example as a proof-of-concept of real-time enhancements made to GPenSIM. The example discussed in this section is purposely made simple for that the details of modeling, simulation, and solving techniques of GPenSIM becomes apparent. This section develops a Petri net model for the well-known traffic signal example, together with control mechanism for pedestrian crossing.

A. *The (Norwegian) Traffic Signal*

Operation of a single independent Norwegian traffic signal with pedestrian crossing:

Normal traffic signal cycle: Let's assume that the signal starts with RED color. Then it turns into RED and YELLOW, and then into GREEN, then follows YELLOW and finally back to RED again.

Pedestrian presses the STOP button: When pedestrian presses the STOP button, then the traffic signal will continue until the RED color becomes active. And then, the traffic signal will stop at RED for a while, to allow the pedestrians to cross, and then resumes the normal cycle.

Emergency button is pressed: If the emergency button is pressed (presumably, the emergency button is not accessible to pedestrians, but remotely controlled by traffic controllers), then the normal operation of the traffic signal will stop immediately turning off all the color lights, except the YELLOW light, which will start blinking. Normal traffic signal cycle can only be started, remotely, by pressing the START button – here again, it is assumed that the activation of the START button is done remotely by the traffic controllers.

B. *The Model*

Figure 7 shows the five modules that make up the Petri net model. In the model, transitions shown with single slab means they are simple and do not use pre-processors and postprocessors; whereas transitions shown with three slabs means they are do call their preprocessors first and they fire for a deterministic time and after completion of the firing, they call their postprocessors.

Module M1- Normal cycle: This module consisting of 4 transition-place pairs for the 4 color sequences. Table 1 explains the functions of preprocessors, and postprocessors of each transition.

Table 1. Functions of a transitions together with its preprocessor and postprocessor during the normal cycle

	preprocessor	***T***	***postprocessor***
Seq-1	***tR_pre***: (RED) is ON	***tR***: 5 sec	***tR_post***: (RED) is off

Seq-2	***tYR_pre***: (RED and YELLOW) is ON	***tYR***: 2 sec	***tYR_post***: (RED and YELLOW) is off
Seq-3	***tG_pre***: (GREEN) is ON	***tG***: 5 sec	***tG_post***: (GREEN) is off
Seq-4	***tY_pre***: (YELLOW) is ON	***tY***: 2 sec	***tY_post***: (YELLOW) is off

The table 1 above explains, for example, that in sequence-1 of the normal cycle, the transition tR (meaning transition for the RED light) first run the preprocessor tR_pre which turns on the RED light. Then the transition starts firing for 5 seconds, which is basically a delay of five seconds. After this firing, the transition tR completes its operation by running its postprocessor tR_post, which turns off the RED light. This sequence thus effectively turns on the RED light for just 5 seconds.

The normal cycle also includes interruption for pedestrian crossings, during which RED light will be turned on for a predefined time. However, the control routine for the interruption to allow pedestrian crossing and resumption of normal cycle is coded within the 'Model M4: Pedestrian Crossing Button Pressed'.

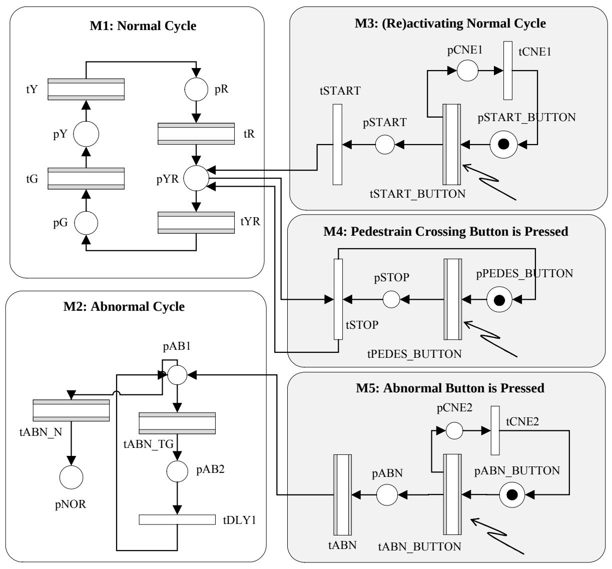

Fig. 7. The Petri net model of the traffic signal problem

Module M2 - Abnormal cycle: This sequence is known as "Traffic signal preemption", where the normal traffic cycle is interrupted giving priority to special traffic (such as emergency traffic for transporting fire engines, ambulances, and police squad cars, though sometimes mass transit vehicles including buses) [13; 14]. The normal cycle interruption is done by remote transmitters that send remote signals (e.g. radio waves, infrared signals, or strobe light signals) that are received by a sensor on or near the traffic lights [14].

During the abnormal cycle, the YELLOW light blinks; this is realized by combination of the transitions tABN_TG and tDLY1: the preprocessor of tABN_TG turns the YELLOW light on, followed by tABN_TG firing for one sec, and then the postprocessor of tABN_TG turns off the YELLOW light. Firing of transition tDLY1 for one second effectively keeps the YELLOW light off for one second. The normal traffic signal cycle can be restored again by remote signal.

Module M3 – (Re)activating Normal Cycle: This module codes the routine for reactivating the normal cycle after a traffic signal preemption e.g. for emergency traffic. Note that the reactivation is triggered by a remote button action that is an external event and is fed by a sensor into the transition tSTART_BUTTON.

Module M4 – Pedestrian Crossing: This module codes the control routine for pedestrian crossing interruption initiated by pedestrian pressing the button. The button action is an external event that is fed by a sensor into the transition tPEDES_BUTTON.

Module M5 – Abnormal Cycle Control: This module codes the control routine for traffic signal preemption, which is initiated by remote action by traffic controllers. This external button action is fed by a sensor into the transition tABN_BUTTON.

The following two subsections (C and D) presents traffic control using two different hardware devices:

1. NI-DAQ Device
2. Lego NXT Mindstorms Robot

Due to brevity, program codes for the two simulations are not shown in this paper. Interested reader is encouraged to inspect the complete code at [15; 20], to enjoy the compactness and the easiness of the program code.

C. Controlling Traffic Lights via NI-DAQ Device

The test-bed for testing Real-Time GPenSIM consists of the following hardware (see figure 8):

- National Instrument NI USB-6525 data acquisition device (DAQ) is used for receiving input signals from the three buttons, and also for operating (outputs) the three lights RED, YELLOW, and GREEN.
- MathWorks Data Acquisition Toolbox (version R2011b) was used to communicate with the NI USB-6525 hardware.
- Custom made circuit board with four buttons (ON/OFF buttons) and with four LED lights.

GPenSIM program code for traffic signal simulation and control using via NI USB-6525 data acquisition device (DAQ), is simple and easy to study. The code is not shown here, but fully available at [15].

Experiment results: There wasn't any remarkable result obtained as the traffic signals performed just as they should be. The only observation that could be mentioned is the speed difference between DAQ device and the simulator: During the sequence-1 of the traffic signal normal cycle (see table 1), RED and YELLOW lights are supposed to be turned ON at the same time during the start of the sequence and then to be turned OFF, again at the same time, at the end the sequence. Though simulator (through tR_pre and tR_post) sent instructions to actuator, the DAQ device sometimes collapses as it cannot update two outputs instantaneously. There must be a delay induced between activating RED and YELLOW lights; but this delay is so small (about 30ms) and hence unnoticeable for human eyes.

D. Controlling Traffic Lights via LEGO Mindstrom Robot

System Requirements (see also [16-19]):

- Operating system: Windows XP
- MATLAB Version 7.7 (R2008b) or higher
- The RWTH–Mindstorms NXT Toolbox
- LEGO Mindstorms NXT building kit 2.0; LEGO Mindstorms NXT firmware v1.26 or higher
- For Bluetooth wireless connection: Bluetooth 2.0 adapter recommended by LEGO

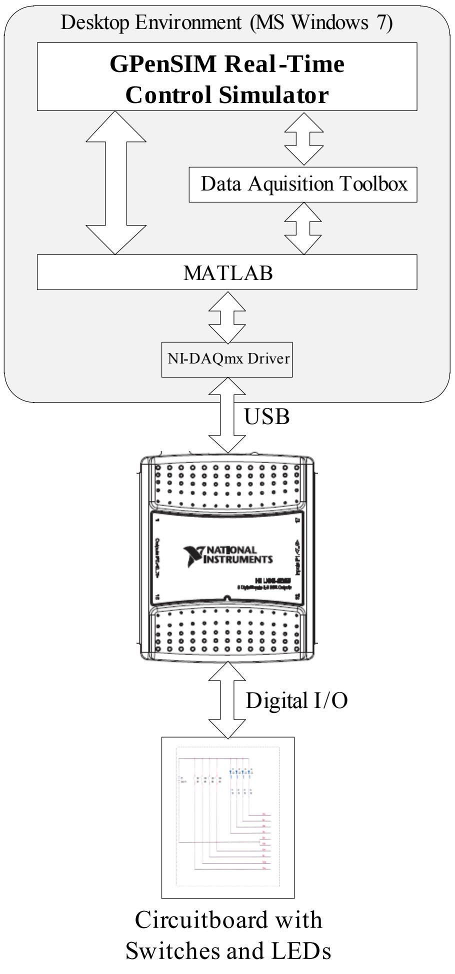
Desktop Environment (MS Windows 7)
GPenSIM Real-Time Control Simulator
Data Aquisition Toolbox
MATLAB
NI-DAQmx Driver
USB
NATIONAL INSTRUMENTS
Digital I/O
Circuitboard with Switches and LEDs

Fig. 8. Real-Time GPenSIM simulating Norwegian Traffic Signal

- For USB connections: Mindstorms NXT Driver "Fantom" v1.02

The RWTH–Mindstorms NXT Toolbox [19] is a software package available for the MATLAB environment that allows control of LEGO Mindstorms NXT robots from a PC; a LEGO robot can be controlled from a PC either through USB connection or through Bluetooth wireless connection.

By using GPenSIM on top of RWTH–Mindstorms NXT Toolbox, various discrete control applications (especially, supervisory control applications) can be developed using LEGO robot as a specimen. Figure-9 shows the various layers of software that can be used to develop applications for discrete robotic control.

Figure-10 shows the LEGO Mindstorms NXT robot setup as a Norwegian Traffic signal, possessing the traffic lights red, green, and yellow, in addition to the three push buttons for pedestrian crossing request, start of emergency cycle, and reset button (for restarting normal traffic cycle).

GPenSIM program code traffic signal simulation and control using LEGO robot, is simple and easy to study. The code is not shown here, but fully available at [20].

Experiment results: The traffic signal simulation and control performed as it should be. However, there were two notable observations from this experiment:

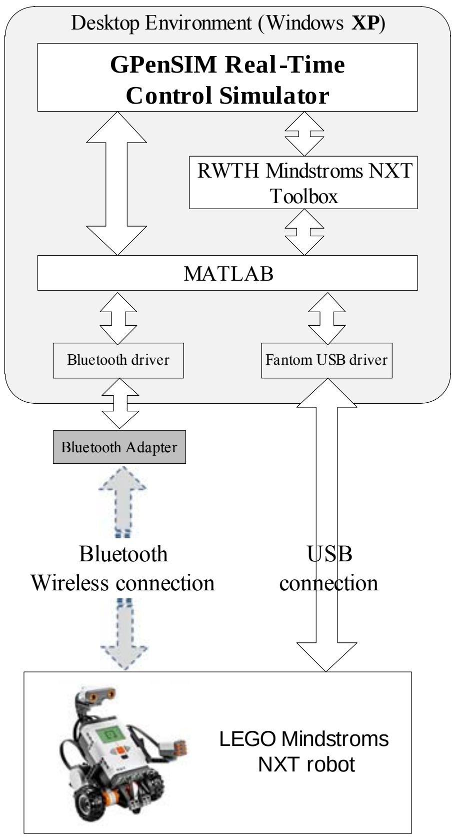
Desktop Environment (Windows XP)
GPenSIM Real-Time Control Simulator
RWTH Mindstroms NXT Toolbox
MATLAB
Bluetooth driver
Fantom USB driver
Bluetooth Adapter
Bluetooth Wireless connection
USB connection
LEGO Mindstroms NXT robot

Fig. 9. Discrete control of LEGO robot with RWTH-Mindstorms NXT Toolbox

Controlling LEGO Mindstorms NXT robot by RWTH–Mindstorms NXT Toolbox, through Bluetooth wireless connection is slow and unreliable; RWTH–Mindstorms NXT Toolbox user manual agrees with this observation [18].

However, controlling the robot by RWTH–Mindstorms NXT Toolbox ***through USB connection is highly reliable***.

Windows *7* Operating System does not allow usage of RWTH–Mindstorms NXT Toolbox for controlling the LEGO robot; Windows 7 blocks the toolbox and MATLAB using either USB or Bluetooth wireless connection to LEGO robot. This problem is not reported in the user manual for RWTH–Mindstorms NXT Toolbox. However, Windows XP works fine with RWTH–Mindstorms NXT Toolbox.

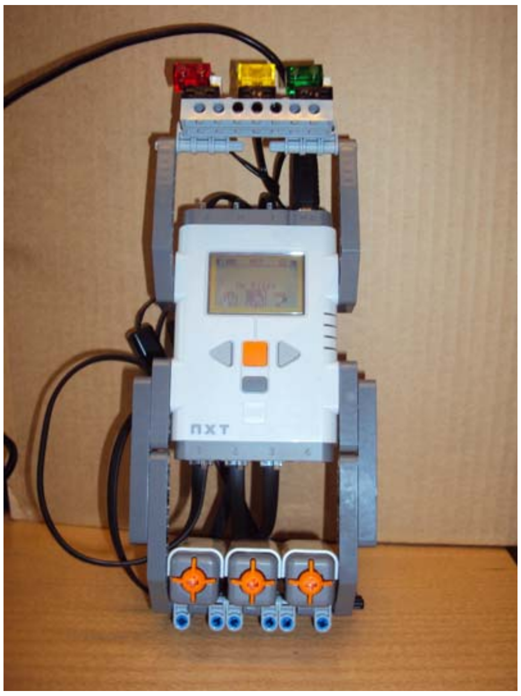

Fig. 10: LEGO Mindstorms robot setup as a traffic lights simulator

VII. CONCLUSION

Petri net is a very powerful tool for modeling and simulation of discrete event dynamic systems (DEDS); Literature study provides many hundred works using Petri nets for modeling and simulation of DEDS. GPenSIM is a simulator based on Petri nets, which has been as a tool to teach discrete simulation, and also been used to solve some industrial problems. This paper talks about enhancing GPenSIM with real-time attributes so that it could function as a real-time control simulator. The application example provided in this paper shows that Real-Time GPenSIM is capable of interacting with external environment, perceive inputs from sensors and triggering output actuators. But the beauty of Real-Time Petri nets generally, and real-time GPenSIM specially is that the control policies that are made is the model is overwhelmingly simple, easy to understand, debug and extend.

REFERENCES

[1] GPenSIM. Available: http://www.davidrajuh.net/gpensim/

[2] G. Cassandras, and S. LaFortune. Introduction to Discrete Event Systems. Hague, Kluwer Academic Publications, 1999

[3] B. Hruz and M. Zhou. Modeling and Control of Discrete-event Dynamic Systems: with Petri Nets and other Tool. Springer-Verlag, London, 2007

[4] R. Davidrajuh. “Distributed Workflow based Approach for Eliminating Redundancy in Virtual Enterprising”. Journal of Supercomputing, Online First, 2011-01-05, DOI: 10.1007/s11227-010-0544-6

[5] R. Davidrajuh and B. Lin. “Exploring Airport Traffic Capability Using Petri Net based Model”. Expert Systems With Applications (ESWA), 38 (2011) pp. 10923-10931

[6] R. Davidrajuh. “Representing Resources in Petri Net Models: Hardwiring or Soft-coding?”. 2011 IEEE International Conference on Service Operations and Logistics, and Informatics, Beijing, China, July 10-12, 2011; pp. 62-67

[7] R. Davidrajuh. “Scheduling using Activity-based Modeling”. 2011 IEEE Conference on Computer Applications & Industrial Electronics (ICCAIE 2011), Penang, Malaysia, December 4-7, 2011

[8] MATLAB. Available: http://www.mathworks.com

[9] S. Haddad. “Time and Timed Petri Nets”. DISC School on Control of Discrete-Event Systems: Automata and Petri nets perspectives, June 6-10, 2011, Cagliari, Italy

[10] B. Berthomieu, and M. Diaz. “Modeling and verification of time dependent systems using Time Petri Nets”. IEEE Transactions on Software Engineering, 17(3), pp. 259-273, March 1991

[11] B. Berthomieu, and M. Menasche. “A State enumeration approach for analyzing time Petri nets”. 3rd European Workshop on Petri Nets, Varenna, Italy, 1982Sfsfsfs

[12] K. Ohashi and K. Shin. “Model-based control for reconfigurable manufacturing systems”. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Seoul, Korea, 2001; pp. 553 – 558

[13] United States DOT Federal Highway Administration (2009) Manual on Uniform Traffic Control Devices, 2009 Edition

[14] United States Patent and Trademark Office. Emergency vehicle traffic signal preemption system. Retrieved October 7, 2005

[15] Traffic Signal Model - Complete Code for NI-DAQ experiment (2012) Available: http://davidrajuh.net/gpensim/2012-IJSSST-traffic-signal-example-NI-DAQ

[16] A. Behrens, L. Atorf, R. Schwann, J. Ball'e, T. Herold, and A. Telle. “First Steps into Practical Engineering for Freshman Students Using MATLAB and LEGO Mindstorms Robots”. Acta Polytechnica Journal of Advanced Engineering, Vol. 48, No. 3, pp. 44--49, June, 2008

[17] A. Behrens, L. Atorf, R. Schwann, B. Neumann, R. Schnitzler, J. Ballé, T. Herold, A. Telle, T. Noll, K. Hameyer, and T. Aach. “MATLAB Meets LEGO Mindstorms - A Freshman Introduction Course Into Practical Engineering”. IEEE Transactions on Education, Vol. 53, No. 2, May 2010

[18] RWTH- Mindstorms NXT Toolbox User Manual v4.04 (2010). List of functions.

[19] RWTH–Mindstorms NXT Toolbox: Available: http://www.mindstorms.rwth-aachen.de

[20] Traffic Signal Model - Complete Code for LEGO Minstorms NXT experiment (2012) Available: http://davidrajuh.net/gpensim/2012-IJSSST-traffic-signal-example-LEGO-NXT