25

GPenSIM: A New Petri Net Simulator

Reggie Davidrajuh
University of Stavanger
Norway

1. Introduction

Petri net is being widely accepted by the research community for modeling and simulation of discrete event-driven systems, mainly due to Petri net's rigorous modeling techniques. There are a number of Petri net tools available for free academic use; see PNWorld (2009) for a list of tools. These tools are advanced tools flexible enough to model complex and large systems. This paper talks about developing a new Petri net simulator. The reasons for building a new simulator are:

- Flexible: the simulator should enable easy integration with other libraries and tools, so that developing hybrid models (e.g. Fuzzy Petri nets, by integrating Petri net with Fuzzy Logic) becomes easy
- Extensible: the simulator should enable users writing their own extensions, either extending or rewriting the existing functions or developing new functions.
- Easy of use: for those who doesn't want to use mathematics when developing a model, the tool should provide a natural language user interface, so that the mathematical details are abstracted away from the user.

General-purpose Petri net simulator (GPenSIM, 2009) is developed by the first author of this paper, in order to satisfy the three criteria stated above (flexible, extensible, and ease of use). GPenSIM is realized as toolbox for the MATLAB platform, so that diverse toolboxes that available in the MATLAB environment (e.g. Fuzzy Logic Toolbox, Control Systems Toolbox) can be used in the models that are developed with GPenSIM.

2. Existing Tools for Discrete Event Simulation

Many tools satisfy some of the three criteria mentioned above. Automata, Stateflow, and Petri nets are the well-known tools used for simulation of discrete event systems. Though automata have a strong footing in computer science, the serious shortcoming with it is the lack of structure – the ability to modularize a system (decompose a system into modules) [2]. Stateflow is commercial software that runs in MATLAB environment [8]. Stateflow is similar to Petri net; converting a Petri net model of a discrete event system into a Stateflow model and vice versa is easy. However, learning Stateflow, with its syntactic, semantic, and graphical details, is much more difficult than learning Petri net. In addition, Stateflow also demands some knowledge of Simulink, in addition to MATLAB.

Petri net is being widely accepted for modeling and simulation of discrete event systems and there is a number of Petri net tools available free-of-charge for academic usage (PNWorld, 2009). These tools are sophisticated tools flexible enough to model complex and large systems. However, these tools are stand-alone systems, and for integrating the functions of these tools with other tools or libraries, one need to program in either high-level languages like Java or C++, or use XML as an intermediary. Thus seamless integration of these Petri net tools with other types of tools (e.g. Control Systems) is not possible.
GPenSIM, written in MATLAB language, allows seamless integration with the other toolboxes that also available in the MATLAB environment. Programming in MATLAB Language is also extremely easy as the language resembles the BASIC language.

3. Architecture of GPenSIM

GPenSIM is designed using the well-proven paradigms in software engineering such as: layered architecture, modular components, and natural language interface.

3.1 Layered architecture

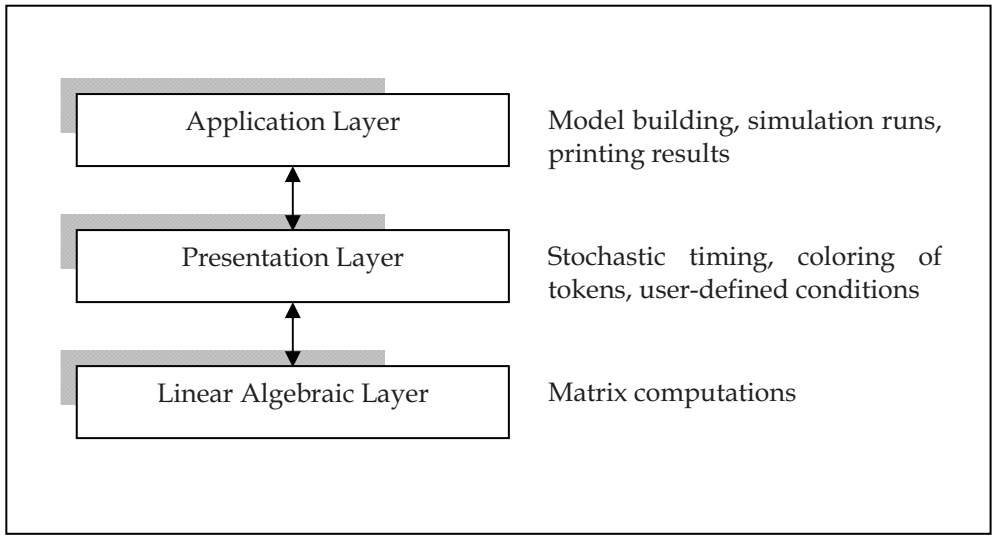


Fig. 1. 3-layer architecture

GPenSIM is built following 3-layer architecture; see figure 1. The bottom layer deals with Petri net run-time dynamics; this layer computes newer states with the help of linear algebraic equations and matrix manipulations. The middle layer adds more high-level functionality such as stochastic timing, coloring of tokens, user-defined conditions ('guard-conditions' in some literature), etc. The top layer offers applications such building a Petri net based model, running simulations, determining coverability tree, printing the simulation results, etc.

3.2 Modular components

A model of a discrete event system developed with GPenSIM consists of a number of files. The main simulation file (MSF) is the file that will be run directly by the MATLAB platform. In addition to the main simulation file, there will be one or more Petri net definition files (PDFs); definition of a Petri net graph (static details) is given in the Petri net Definition File. There may be a number of PDFs, if the Petri net model is divided into many modules, and each module is defined in a separate PDF. While the Petri net definition file has the static details, the main simulation file contains the dynamic information (such as initial tokens in places, firing times of transitions) of the Petri net. In addition to these files (main simulation file and Petri net definition files), there can be a number of transition definition files (TDFs) too.

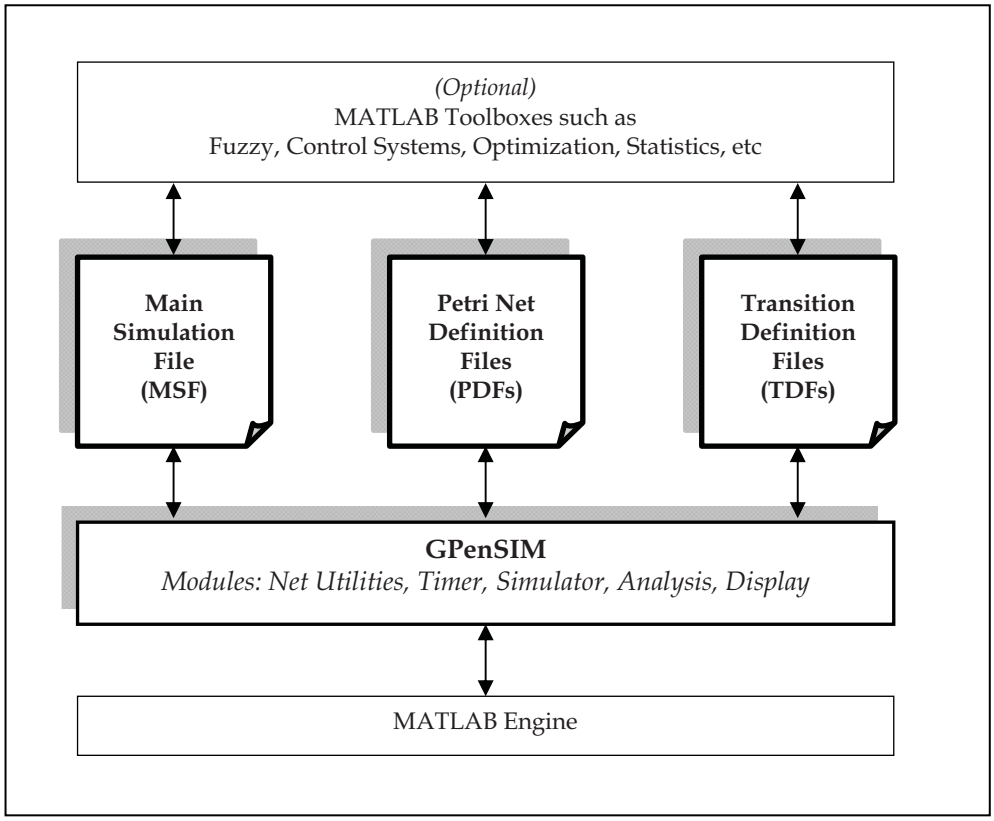


Fig. 2. The architecture of GPenSIM

A transition definition file consists of additional conditions that determine whether an enabled transition can fire or not. The additional conditions are called 'user defined condition' in GPenSIM terminology, whereas in some other literature (e.g. Colored Petri Net (CPN)) it is referred to as 'guard-functions'). There can be a separate transition definition file for each transition in a Petri net model.

3.3 Natural language interface

Users need not know Petri net mathematics when creating a Petri net model of a discrete event system. GPenSIM offers a natural language interface with which model building mainly deals with identifying the basic elements of a system and establishing the connections between these elements. Figure 2 shows the overall architecture of GPenSIM.

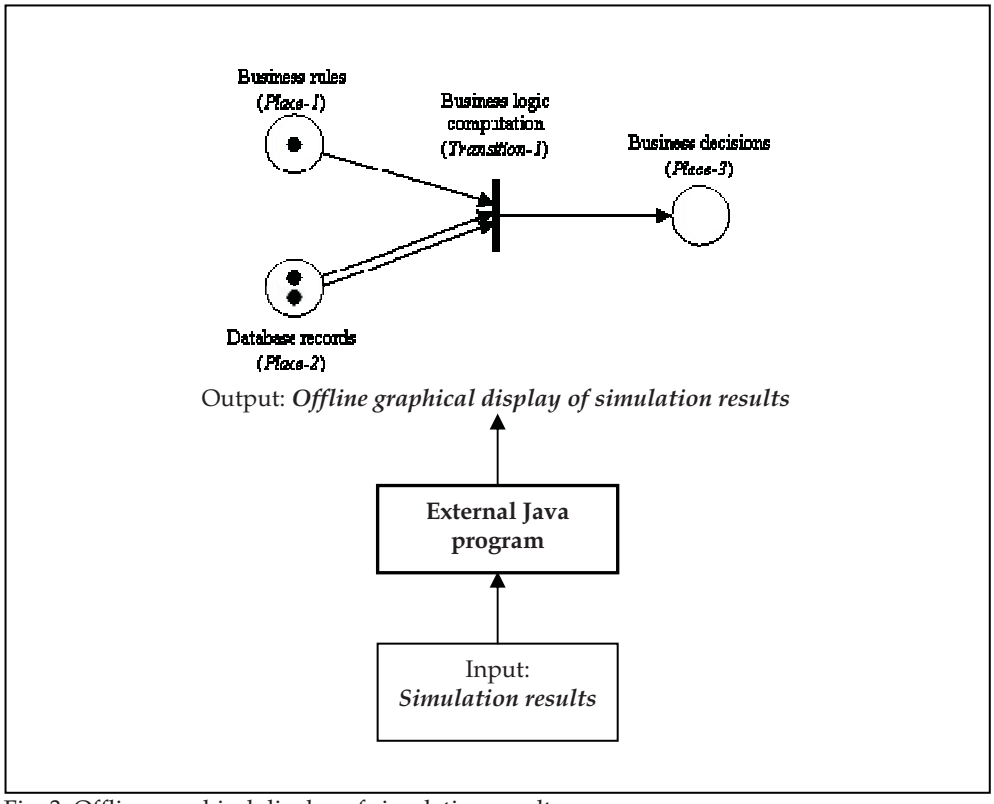


Fig. 3. Offline graphical display of simulation results

3.4 Offline graphical display

After simulation runs, the simulation results can be used for printing results both in ASCII and in graphic format. The results can be also used for off-line (non-interactive) graphical display of step-by-step simulation run; to do the offline display, we need an external program, written in high level language like Java or C#. At present, an external Java based program is under construction. However, step-by-step online (interactive) monitoring of simulation run in progress is neither available at present nor planned for construction in the near future.

3.5 The main loop

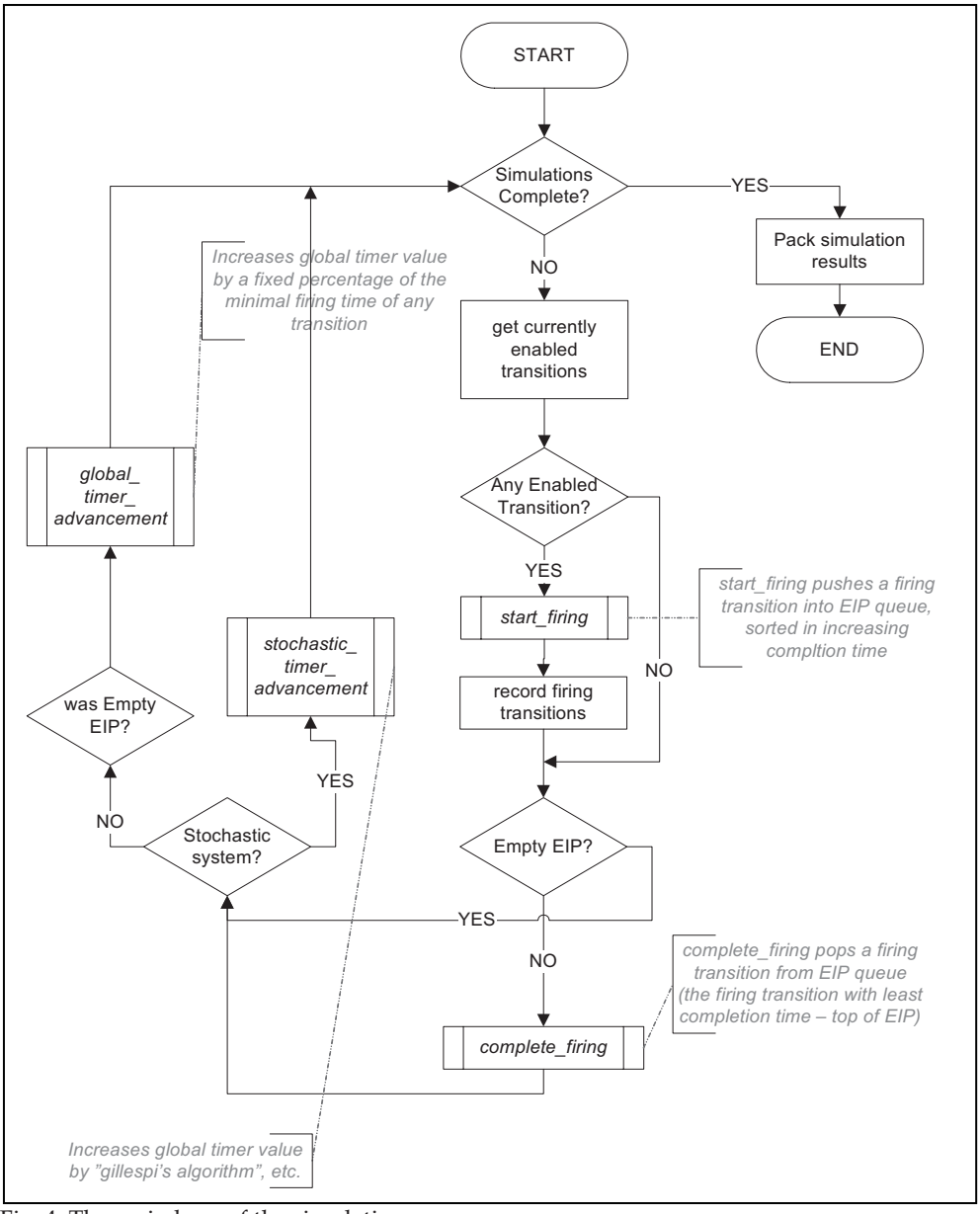


Fig. 4. The main loop of the simulation runs

Figure 4 shows the main loop of the simulator. As in any Petri net simulator, the main loop consists of a simple cycle that first checks whether any transitions are enabled and then it

puts the enabled transitions into the firing queue, provided that the transitions satisfy additional user defined conditions, if any; inputs tokens are also taken away (consumed) by the corresponding transitions. Then, the loop checks whether any firing transitions are completing or have completed. In this case, the firing transitions are popped out of the firing queue and output tokens are deposited into the respective output places.

Figure 4 also shows that there are two kinds of timers are in use. The first timer – called global timer is the one that is normally used. The second timer called stochastic timer is used only for 'stochastic systems'. Stochastic systems can be leisurely defined as continuous systems (as opposed discrete systems) that are to be discretized first into discrete systems so that a Petri net model can be created for them. A case study on stochastic system is done in section 5.

Finally, the main loop shown in figure 4 hints an extension to GPenSIM: by using a Real-Timer (computer's real-time clock) instead of stochastic or global timer, a Real-Time GPenSIM version can be developed. This Real-Time GPenSIM is basically a soft Programmable Logic Controller (PLC), which will use a Digital & Analogue Input Output Card (DAC) to read sensor inputs from the outside world and will also output digital signals to triggers via the card. In this real.-time version, the main loop should read the sensor data at the start of each cycle, and the state of the firing transitions should be mapped to the output triggers.

4. Methodology for Modelling and Simulation with GPenSIM

Creating a Petri net model consists of two steps:

1) Defining the static Petri net graph, and
2) Assigning initial dynamics in the main simulation file

Step-1) Defining the Petri net graph in one or more Petri net Definition Files (PDF): this is the static part. This step consist of three sub-steps:

   a. Identifying the basic elements of a Petri net graph: the places,
   b. Identifying the basic elements of a Petri net graph: the transitions, and
   c. Connecting the elements with arcs

Step-2) Assigning the dynamics of a Petri net in the Main Simulation File (MSF):

   a. The initial markings on the places, and possibly
   b. The firing times of the transitions

After creating a Petri net model, simulations can be done.

5. Application Example

GPenSIM has been used for modeling different types of discrete event system; e.g. Davidrajuh (2007) presents model of an adaptive supply chain; Davidrajuh (2009) presents a simulation study of a Bluetooth Wireless technology based classroom tool.

This application example deals with discretizing of continuous systems. Generally, Petri net is for discrete event simulations only. However, if a continuous system can be discretized, then this system could also be modeled with Petri nets. However, discretizing of a continuous system is not easy and needs some understanding of Petri net formalism and matrix representation; interest reader is referred to Wilkinson (2006).

The application example is a prey-predator (e.g. rabbit-fox) ecological equilibrium. The equilibrium is stated by 2 simple differential equations (known as Lotka & Volterra equation):

- The specimen prey (e.g. rabbit - r) mutates by itself and depleted by predators (e.g. foxes - f):

$$\frac{dr}{dt} = (\alpha \cdot r) - (\beta \cdot r \cdot f) \tag{1}$$

- The specimen predator (e.g. fox) grows due to rabbits (access to food) and depleted by its own population (competition for food):

$$\frac{df}{dt} = -(\gamma \cdot f) + (\delta \cdot r \cdot f) \tag{2}$$

- $\alpha, \beta, \gamma,$ and δ are parameters representing the interaction of the two species.

The equilibrium is determined by partial differential equations; equivalent Petri net model for the interaction is given in figure 5.

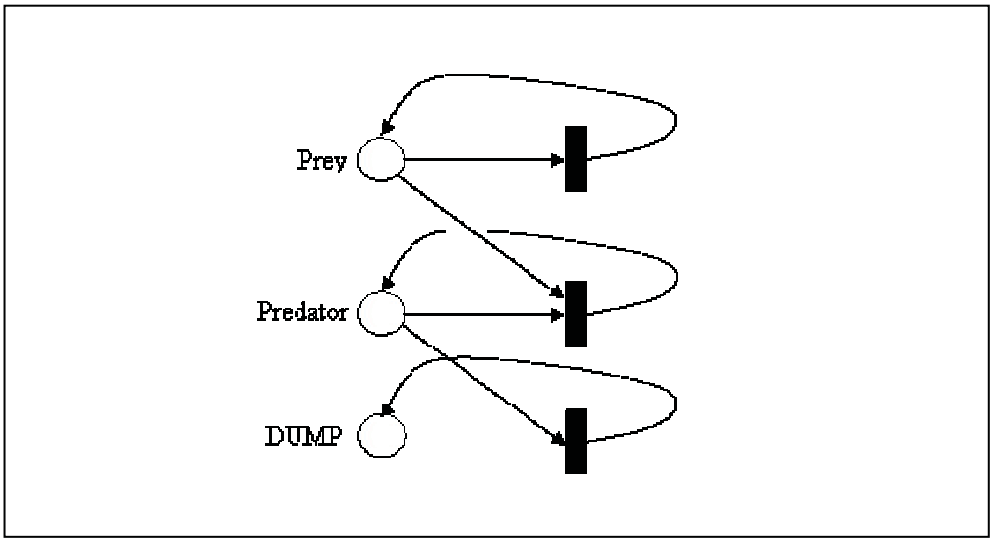


Fig. 5. Petri net model of Pre-Predator interaction

5.1 Creating the model

Petri net definition file (PDF) that defines the static Petri net graph of figure 5 is given below:

```
% function: Petri Net Definition File (PDF)
% filename: predator_prey_def.m

function [PN_name, set_of_places, set_of_trans, set_of_arcs]...
                = predator_prey_def(global_info)
PN_name='predator-prey p/151';
set_of_places = {'Prey', 'Predator', 'DUMP'};
set_of_trans = {'t1','t2','t3'};
set_of_arcs = {'Prey','t1',1, 't1','Prey',2,...
   'Prey','t2',1, 'Predator','t2',1, 't2','Predator',2,...
   'Predator','t3',1, 't3','DUMP',1};
```

The main simulation file (MSF) is give below. MSF first identifies the PDF and then assigns the initial dynamics. Then, it runs the simulations using the function ‘gpensim’. Finally, the simulation results are printed.

```
% function: MAIN SIMULATION FILE (MSF)
% filename: predator_prey.m
global_info.MAX_LOOP = 10000;    % many simulation runs
global_info.c = [1 .005 .6];
global_info.STOCHASTIC = 1; % stochastic timer
pn = petrinetgraph('predator_prey_def');
dynamicpart.initial_markings = {'Prey',50, 'Predator', 100};
sim = gpensim(pn, dynamicpart, global_info);
```

```
M = plotp(pn, sim, {'Prey','Predator'});   %figure 6a
plot(M(:,1), [M(:,2), M(:,3)]);   % figure 6b
```

Stochastic timer: Due to discretization, the simulations should use stochastic clock, rather than the inbuilt global timer Wilkinson (2006). The realization of *Gilespi* algorithm (Gilespi, 1977) for advancing stochastic timer is given below.

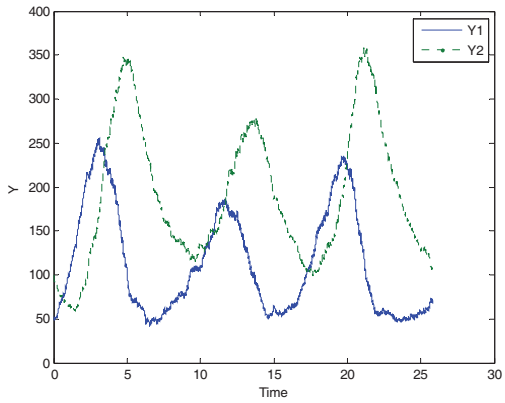
```
% function: realization of Gilespi's algorithm
% filename: time_advancement.m
function [pn, global_info] = time_advancement(pn, global_info)
c1=global_info.c(1);  c2=global_info.c(2);  c3=global_info.c(3);
Prey = get_place(pn, 'Prey');
PRED = get_place(pn, 'Predator');
h1 = c1 * Prey.tokens;
h2 = c2 * Prey.tokens * PRED.tokens;
h3 = c3 * PRED.tokens;
H = h1 + h2 + h3;
%%%%  probabilities
global_info.pro1 = (h1/H);
global_info.pro2 = (h2/H);
global_info.pro3 = (h3/H);
delta_T = 1-exp(-1/H);
pn.current_time = pn.current_time + delta_T ;
```

Finally, Transition Definition File (TDF) for the transition t1 is given below. TDFs for the transitions t2 and t3 are similar.

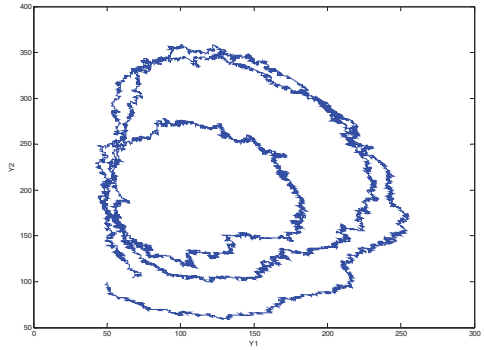
```
%  function: Transition Definition File (TDF) for transition t1
%  filename: t1_def.m
function [fire, new_color, override, selected_tokens,global_info] = ...
        t1_def (pn, new_color, override, selected_tokens,global_info)

c1=global_info.c(1); c2=global_info.c(2); c3=global_info.c(3);
Prey = get_place(pn, 'Prey');
PRED = get_place(pn, 'Predator');
h1 = c1 * Prey.tokens;
h2 = c2 * Prey.tokens * PRED.tokens;
h3 = c3 * PRED.tokens; H = h1 + h2 + h3;
%%%%  probabilities
pro1=(h1/H); pro2=(h2/H); pro3=(h3/H);

R = rand*(1);
fire =  (R <= pro1);
```

5.2 Simulation results

Figure 6a shows variation of prey and predator population with time. Figure 6b shows how the prey population varies against the predator population (prey-predator equilibrium).



6a) Composition of specimens Prey-Predator with time



6b) Prey-Predator equilibrium

Fig. 6. Simulation results

6. Conclusion

This chapter presents a new Petri net simulator, called General Purpose Petri Net simulator (GPenSIM), for modeling and simulation of discrete event systems. The tool is devised to achieve the following:

- Flexibility: ability to cooperate with diverse tools and libraries
- Extensibility: ability to offer support for rewriting or extending existing functions or new functions
- Ease of use: Tool may be based on rigorous mathematics, but the user need not know it

At present, GPenSIM has the following limitations:

- It is based on a commercial platform (MATLAB), which is not free for academic (or commercial) use.
- GPenSIM does not incorporate online (interactive) simulator. Thus, monitoring the system during the simulation run is not possible. A Java based program for offline graphical display of the simulation results is under construction.
- Though GPenSIM offer extensibility, it comes with a cost: one need to program in MATLAB language. Though programming in MATLAB is easy as this language resembles a simpler BASIC language, still one need to spend some time to learn the language.

Further Work:

There are numerous possibilities for extending GPenSIM. We give blow just two:

- Adaptive GPenSIM: a version of GPenSIM in which the arc weights are not fixed and can vary during the simulation run.
    - Self adaptive: In each TDF, the arc weight of the transition can be changed.
    - Forced adaptive: in a specific TDF, arc weights of any transition can be varied
- Real-time ("soft PLC") simulator: Instead of global timer, the real-time clock of the computer can be used. In this case, the GPenSIM is no longer just a simulator, but it becomes a soft Programmable Logic Controller.

7. References

Cassandras, G. and LaFortune, S. (1999) *Introduction to Discrete Event Systems*. Hague, Kluwer Academic Publications

Davidrajuh, R. (2007). "A Service-Oriented Approach for Developing Adaptive Distribution Chain", *International Journal of Services and Standards*, Vol. 3, No.1, pp. 64 – 78

Davidrajuh, R. (2009)."Evaluating Performance of a Bluetooth-based Classroom Tool". *International Journal of Mobile Learning and Organisation,* Vol. 3, No. 2, pp. 148-163

Extend (2009). Available: http://www.imaginethatinc.com/

Gilespi, D. (1977) "Exact Stochastic Simulation of Coupled Chemical Reactions". *The Journal of Physical Chemistry*, Vol. 1, No. 25, pp. 2340 – 2351

GPenSIM (2009). Available: http://www.davidrajuh.net/gpensim/

LabView (2009). Available: http://www.ni.com
MATLAB (2009). Available: http://www.mathworks.com
Petri net world (2009). Available:
http://www.informatik.uni-hamburg.de/TGI/PetriNets/
Pritsker Corporation (1990). SLAM II Quick Reference Manual. Pritsker Corporation, West Lafayattee, IN, USA
SIMCSRIPTII (2009). Available: http://www.simscript.com/
Wikipedia (2009). Available: http://www.wikipedia.org
Wilkinson, D. (2006) *Stochastic Modelling for Systems Biology*. Chapman & Hall / CRC, NY