Laboratory 8

Real-Time Java – Object Enhanced Real Time Petri Nets (OER-TPN)

1. Laboratory objectives

- Familiarity with the specifications and the *Java framework of Object Enhanced Real Time Petri Nets* (*OER-TPN*);
- Implementation of *real-time Java* applications.

2. Getting started

Java is an object-oriented programming language developed by *Sun* (later purchased by Oracle). *Java* programs are executed (interpreted) by another program called *Java Virtual Machine* (*Java VM*). So *Java* programs do not run directly on the operating system, but they need Java *VMs* to be interpreted.

A *Java* program consists of a collection of classes in one or more source files with the *.java* extension. The programs are compiled (using the *javac.exe* utility) and the *byte code* is generated so that for each class that is defined in the source files, a file with the .class extension will be generated. The compiled files contain *JVM* instructions. To run the program you need a *Java virtual machine* that interprets the intermediate code according to the platform (operating system) where it runs. Hence, perhaps the most important advantage of *Java* applications is code portability.

Therefore, the *Java virtual machine* reads the classes that compose the application and translates them into instructions that are specific to the platform where the program runs.

2.1 Java Virtual Machine Architecture

Figure 8.1 shows the *Java virtual machine* architecture. It contains a subsystem that has the role of loading application classes into memory. Also, every *Java virtual machine* has a component called *Execution Engine*, a mechanism responsible for executing the instructions in the classes loaded into the system.

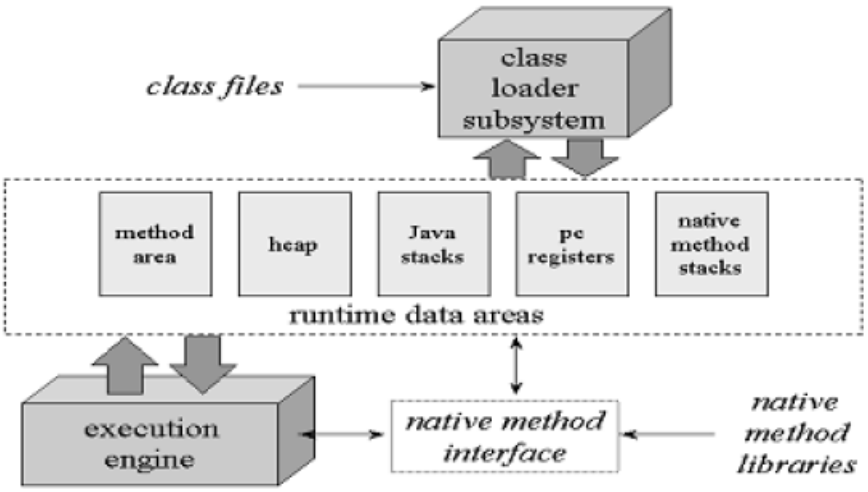


Figure 8.1 Java Virtual Machine Architecute

In addition to the two subsystems loading classes and executing instructions, JVM also contains a data area. It is divided into the following components:

- *PC Registers* (*program counter*): contains the address of the instructions running in the *JVM*. If the instruction being executed is native, then this register is undefined.
- *Method area*: When a class is uploaded to the system, *JVM* analyzes the information about the class structure and uploads this data to the *method area*.
- *Heap*: When the programs are executed, all built objects are placed in the *heap*. The *method area* and *heap* are shared by all threads of execution running within the virtual machine.
- *Java stack:* Each running thread has a private area in the *Java* stack. In this area, the instruction executed by each thread (execution) is identified.
- *Native method stack*: This area identifies the native instruction that is running.

2.2 Java in real-time applications

Standard *Java* implementations are not suitable for real-time application development for the following reasons:

- The way in which *Java* threads of execution are scheduled for execution is not fully specified in order to make the development of virtual machines (*JVM*) easier for new platforms;
- The *Java Garbage Collector (GC)* mechanism has higher execution eligibility than any other thread, which can lead to unexpected delays in thread execution (which awaits *GC* execution);
- The *heap* area where objects are loaded is under *GC* control - for real-time applications it is necessary to have memory areas independent of any automatic memory release mechanism.

In order to solve this problem, *Real-Time Java Experts Group* has defined and published *Real-Time Specifications for Java* (*RTSJ*) (http://rtj.org) which provides for the following:

- A new model for memory management;
- Direct access to physical memory;
- Greater time granularity, more suitable for real-time applications;
- Increased accuracy of how threads are scheduled for execution;
- Mechanisms for handling asynchronous events;

The *Real-Time Specifications for Java* programming model is shown in Figure 8.2.

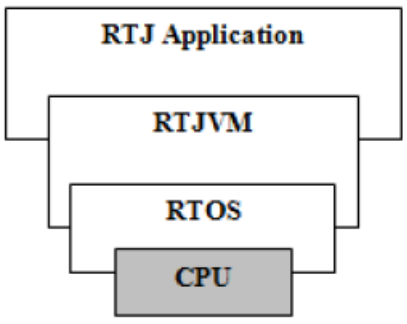


Figure 8.2 RTSJ programming model

Currently there are several implementations of these specifications including: *TimeSys RTSJ Reference Implementation, Aicas Jamaica, jRate*. *The Aicas JamaicaVM 3.0* distribution will be used within the Real Time Systems laboratory.

2.3 Object Enhanced Real Time Petri Nets (OER-TPN) Theory

The Object Enhanced Real-Time Petri Net (OER-TPN) that is a modification of the previous OETPN (Object Enhanced Time Petri Net) [1]. The modification allows the adding of new features for:

- describing moving objects that carry temporal information related to time such as speed, acceleration or deceleration,
- modeling the changing of state involved by dynamically created temporal information and static temporal information,
- describing of activities having associated deadlines and _ detecting the missing deadline.

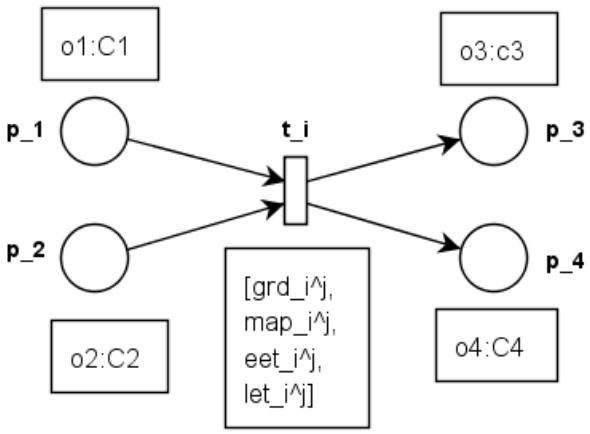


Figure 8.3 Example of an OER_TPN model

The OER-TPN models are obtained from the Object Enhanced Time Petri NETs (see [1]) endowing them with variable time intervals for the transition executions. The earliest and latest execution time of transitions can be calculated from information stored in the tokens of the transition activated input places. This information related to timed behavior (such as speed, acceleration, deceleration, etc.) allow the specification and verification of R-T requirements or properties. An example of an OER–TPN model is presented in Figure 8.3. The following notations are used:

- $N = (P, T, F)$ is a net with two kinds of disjoint node sets
- a finite place set $P = \{p_1, p_2, \ldots, p_m\}$, $(m \geq 0)$,
- a finite transition set $T = \{t_1, t_2, \ldots, t_n\}$, $(n \geq 0)$,
- $F \subseteq P \times T \;\; T \;\; P$ flow relation,
- ${}^{o}t = \{p \in P | (p, t) \in F\}$ the transition t input place set
- $t^{o} = \{p \in P | (t, p) \in F\}$ the transition t output place set
- $Inp = \{pi_{1,\ldots}\} \subseteq$ P, are external inputs that inserts tokens into the net,
- $Out = \{po_{1,\ldots}\} \subseteq$ P, are output channels that extracts tokens from the net sending them to channet destination tokens,
- $Types = \{Class_1, Class_2, \ldots\}$ represents the set of software classes of the net,
- *type*: $P \rightarrow Types$ is a mapping that assigns to each place a class (i.e. type), (the mapping is of *type(p)* is a class*)*
- a guard of a transition t_i denoted by grd_i^j is a mapping:

$$grd_i^j : \prod_{p \in {}^o t_i} \{Domain(M(p)) \cup \phi\} \rightarrow \{true, false\}$$

where true and false are the values of the boolean set. ϕ denotes the empty set. The set of all guards assigned to a transition t_i is denoted by grd_i, and **Grd** denotes the set of all the guards of the net.

- A transition t_i mapping denoted map_i^j is :

$$map_i^j : \prod_{p \in {}^o t_i} \{Domain(M(p)) \cup \phi\} \rightarrow \prod_{p \in t_i^o} \{Domain(M(p)) \cup \phi\}$$

- *Domain*(*M*(*p*)) defines for the token that can be set in the place *p* the set (possible very complex and infinite) of the values assignable to the object's attributes. All the mappings of a transition t_i are included in the set map_i and **Map** represents the set of all the mappings of the net.
- a transition t_i earliest execution time (delay) latest execution time (delay) are mappings eet_i^j, let_i^j

$$eet_i^j, let_i^j : \prod_{p \in {}^o t_i} \{Domain(M(p)) \cup \phi\} \rightarrow \mathbb{R}$$

$\mathbb{R}$ means the set of real numbers considering the time as being dense. Often the set $\mathbb{N}$ of the natural numbers is used when the time is accepted as being discrete and it is measured with the clock accuracy. The set of all earliest execution time mappings assigned to a transition t_i is denoted by eet_i, and **Eet** denotes the set of all the delay mapping sets of the net. The set of all latest execution time mappings assigned to a transition t_i is denoted by let_i, and **Let** denotes the set of all the delay mapping sets of the net.

The definition of an OER-TPN is:

OER – TPN = (P, T, pre, post, Inp, Out, D, *δ, Types, type*,

Grd, ***Map***, *Λ, M, init, end)* (1)

Where

- $\Lambda = \{\lambda_t | t \in T, \lambda_t \subseteq grd_t \times map_t \times eet_t \times let_t\}$ is a set of relations $\lambda_t = \{(grd_t^i, map_t^i, eet_i, let_i)\}$ that assigns to each transition *t* tupples composed of a guard, a mapping, an *eet* and *let* from the sets grd_t, map_t, eet_i, and let_i.
- **M** is the net vector marking with *M*(*p*) the marking of the place p that identifies the object token of the type (p).
- *init* is a function that initializes the net state (i.e. the markings) with values from the creators domain and
- *end* is a function that terminates the OET-TPN model execution and provides the creator processed information.

2.4 OER-TPN Framework

The OER-TPN framework shown in Figure 8.4 is implemented based on the execution algorithm [1]:

Input: ***Pre***, ***Post***, M^0, *P*, *T*, *D*, *Grd&Map*, Out, Inp;
Initialization: $M = M^0$; *execList = empty;*
repeat
    wait(event);
    if event is *tic* **then**
        * decrease the Delays of the transitions in *execList*;
    else
        receive(Inp);
        * update **M**;

end if;
repeat
for all t_i *2 T* do
if there is met at least one *grd* in the t_i Grd_i list
for M(p), $p \in {}^{o}t_i$, **then**
* move the tokens of ${}^{o}t_i$ from **M** to M_t;
* add t_i to *execList*;
$Delay[t_i] = \delta(t_i)$;
end if;
end for;
for all t_i in execList do
if ($Delay[t_i]$ is 0) **then**
* remove *ti* from *execList*;
* calculate the tokens for **M** in t_i^o;
* remove the tokens from M_t for all ${}^{o}t_i$;
* set the tokens in $t_i^o$and start the active tokens;
end if
if $t_i \in$ *Out* **then**
send(Out);
end if
end for
until there is no transition that can be executed;
until the time horizon;
END algorithm;

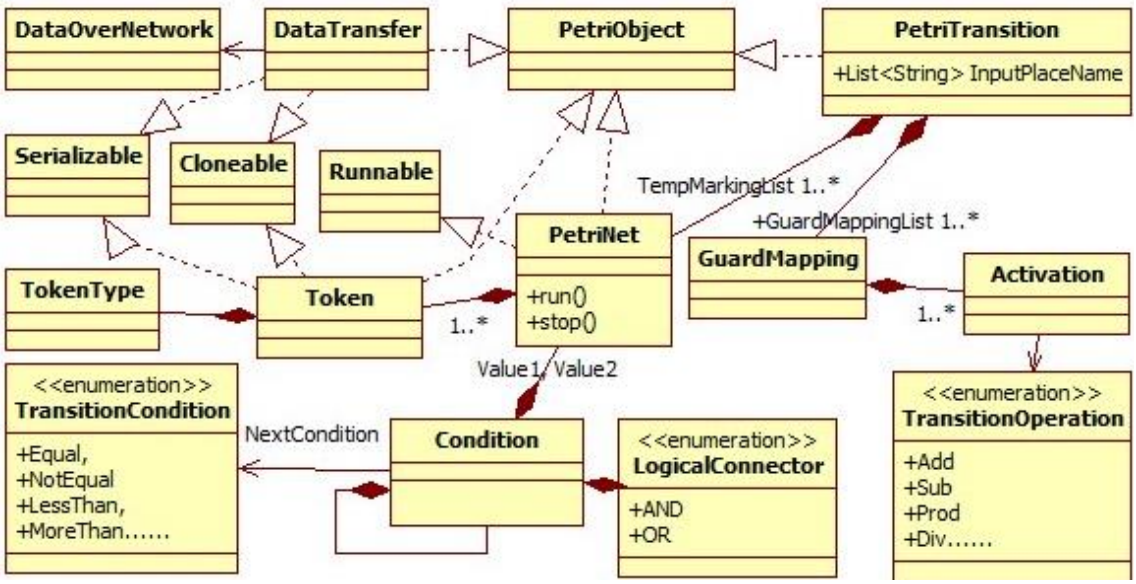


Figure 8.4 OER-TPN Framework

Each Petri net is an object that contains a list of places and a list of transitions. The transition must have a list of GuardMapping containing the guards which is an instance of the Condition class that is connected to the next condition with the LogicalConnector (AND, OR), the Condition class compares between two values (tokens). The mapping is activated when the transition's condition is true, it is an instance of the Activation class that contains the TransitionOperations that are performed on the tokens to calculated the output places of the transition. The marking is a list of places, the place type is either passive (objects), active (sub OER-TPN s), or output channels of type DataTrasnfer. The requirements for creating a new passive token type are, adding transition conditions and transition operations to implement the guards and mappings that are specific for each token type. The newly defined token types must implement the PetriObject interface so they can be recognized as Petri Net objects and treated as such.

3. Example of an Application

Consider the galvanizing line for the ETPN model in the figure 8.5. The mobile robot M takes one container from the conveyor belt R_0, puts it in the hardware resources (basins) R_1, ⋯, R_{n-1} according to the given technology requirements. Finally, the container reaches the conveyor belt R_n and disappears from the installation. The order of resources is given by the current technology.

Variant 1: the travel times of M from one resource to another are known.

Option 2: a resource sensor must be mounted to signal (event) the positioning of M next to the resource.

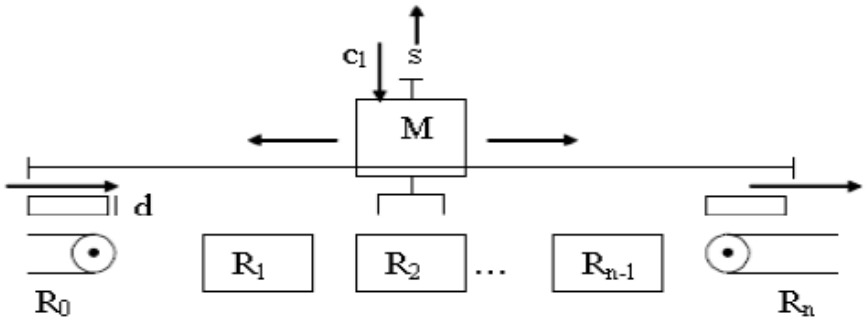


Figure 8.5 galvanizing line

Requirements: It is required to design an OER-TPN model of the installation that is functionally equivalent to the ETPN model (shown in the figure 8.6).

For the OER-TPN model, the following are required: the structure, the types of locations and the evolution relations (grd, map, eet, let) of the transitions.

Evaluation criterion: one OER-TPN model of the installation is better than another (OER-TPN) if it has a smaller number of nodes, although it fully describes the behavior of the installation.

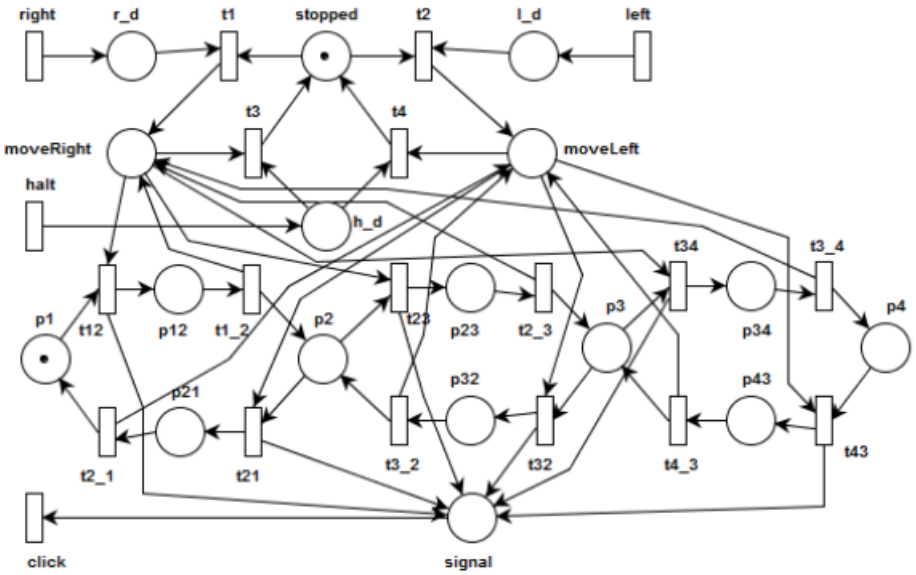


Figure 8.6 ETPN model

Solution:

A. The Controller Model

The OER-TPN model of the controller is shown in figure 8.7. The place types are as follows:

type(p_i1) = type(p_1)= type(p_o1)=type(p_i2)= {R_k; int}
type(p_2) = {move: int}; move = 0 ➔ halt; x=x
move = 1 ➔ right; x=x +1
move = -1 ➔ left; x= x -1

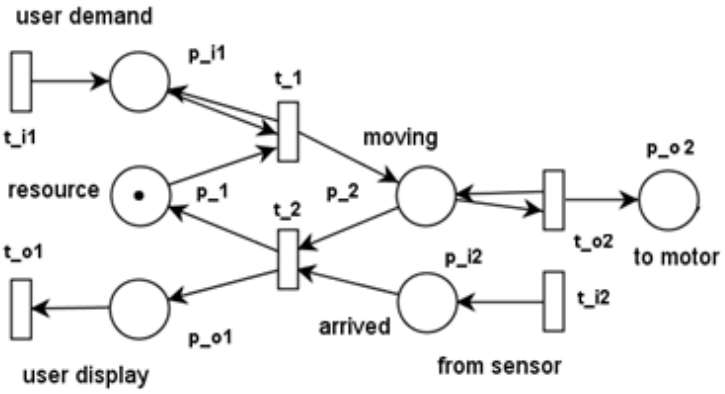


Figure 8.7 The controller OER-TPN model

The Transitions guards and mappings are as follwos:

t_1:
p_i1.x = p_1.y ➔ map(p_2) = 0; map(p_i1) = φ
p_i1.x > p_1.y ➔ map(p_2) = +1;
p_i1.x < p_1.y ➔ map(p_2) = -1;
t_i2 provides the position (i.e. {0, 1, ..., , 10}

t_2:
(m(p_i2) ≠ φ) ➔ map(p_1): p_1.x=p_i2.x
map(p_o1): p_o1.x= p_i2.x

t_o2:
p_2.x = 0 ➔ halt
p_2.x = +1 ➔ right ➔ +1
p_2.x = -1 ➔ left ➔ -1

B. The Robot Model

The OER-TPN model of the robot is shown in figure 8.8. The place types are as follows:
type(p_1) {x:int}= type(p_1) = type(p_o);
Init: p_0.x=0;

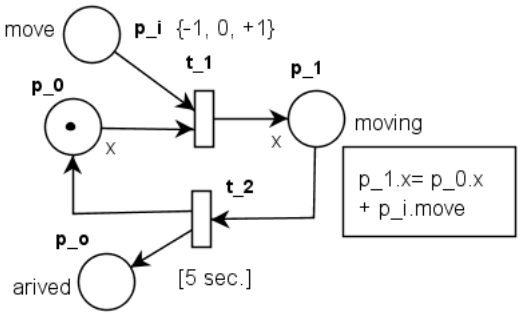


Figure 8.8 The robot OER-TPN model

The Transitions guards and mappings are as follows:

t_1: p_1. x = p_0.x + p_i.move;

t_2: p_o.arived=p_1.x ;
p_0.x = p_1.x;

if it receives o then no 5 sec delay because it doesn't has to move.

C. Implementation with OER-TPN Framework

The implementation is given with the linke below:

https://bitbucket.org/dahliajj/oetpn_oertpn_framework/src/master/

It is an eclipse workspace, in the *OER_TPN_Lab* package. Make sure to navigate to one (NewOETPN folder).

When testing, run the *GUI package/ InputInt* class so we can send input through the keyboard to the input channel (p_i1) of the controller as shown in figure 8.9.

In the first field you have to input the name of the input channel, in the second field is the desired location for the robot, and the last field represents the port number of the OER-TPN which is by default 1080 where in this case is the port number for the controller.

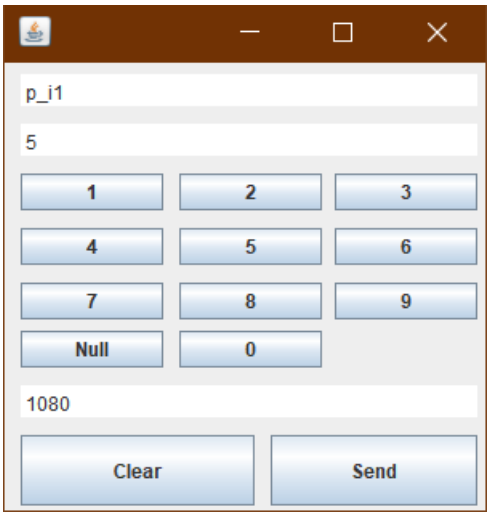


Figure 8.9 The GUI/ Input Int

Code explanation:

In code sequence 1, a *PetriNet* object is created and given a name (Controller Petri) so it can be shown on the *GUI* of the controller.

```
PetriNet pn = new PetriNet();
pn.PetriNetName = "Controller Petri";
```

The places are created according to their data types, for example p_1 is of type *DataInteger*, the place name is considered as the place ID to access the data, The initial value can be set using the *setValue*(*value*) method, then each place must be added to the *PlaceList* of the *PetriNet* object pn.

```
DataInteger p_1 = new DataInteger();
p_1.SetName("p_1");
p_1.SetValue(3);
pn.PlaceList.add(p_1);
```

The Output channels are places with the type *DataTransfer*, in the place value we must introduce the IP address where the targeted OER-TPN is, its port number, and the place ID of its input channel. In our case, the Robot OER-TPN is running on the same computer so the IP is set to the *locahost*, the port number of the *Robot* is 1081, and its input channel ID is p_i.

```
DataTransfer p_o2 = new DataTransfer();
p_o2.SetName("p_o2");
p_o2.Value = new TransferOperation("localhost", "1081", "p_i");
pn.PlaceList.add(p_o2);
```

If we want to compare values in the *Condition* or set values in the mappings of any transition, we need to have a place to compare with or to get the value form. So extra places are defined to store the values and they are not connected to the Petri net. In this example, it is needed to crate Halt = 0, Right = 1, and Left = -1

```
DataInteger Halt = new DataInteger();
Halt.SetName("Halt");
Halt.SetValue(0);
pn.ConstantPlaceList.add(Halt);
```

```
DataInteger Right = new DataInteger();
Right.SetName("Right");
Right.SetValue(1);
pn.ConstantPlaceList.add(Right);

DataInteger Left = new DataInteger();
Left.SetName("Left");
Left.SetValue(-1);
pn.ConstantPlaceList.add(Left);
```

The transitions are defined as *PetriTransition*, the *TransitionName* is the transition ID. The input places for the transition must be added. The variables that are used for comparison or for mapping are added as well.

```
PetriTransition t_1 = new PetriTransition(pn);
t_1.TransitionName = "t_1";
t_1.InputPlaceName.add("p_i1");
t_1.InputPlaceName.add("p_1");
t_1.InputPlaceName.add("Halt");
t_1.InputPlaceName.add("Right");
t_1.InputPlaceName.add("Left");
```

To define a guard for the transition, Condition objects must be created containing the transition ID, the input place, and the condition operation. The *Condition* objects must be linked with *LogicConnector* (AND, OR).

The implementation for guard t_1: $p\_i1 \neq \phi$ AND $p\_1 \neq \phi$ AND $p\_i1 = p\_1$

The second condition is connected to the last one with an AND, the first condition is connected to the second one with an AND, just like a *LinkList*.

```
Condition T1Ct1 = new Condition(t_1, "p_i1", TransitionCondition.NotNull);
Condition T1Ct2 = new Condition(t_1, "p_1", TransitionCondition.NotNull);
Condition T1Ct3 = new Condition(t_1, "p_i1", TransitionCondition.Equal,
 "p_1");

T1Ct2.SetNextCondition(LogicConnector.AND, T1Ct3);
T1Ct1.SetNextCondition(LogicConnector.AND, T1Ct2);
```

Also, a *GuardMapping* list must be created to add the first condition part, and to add the *Activations* (mappings). The Activation object must contain the transition ID, the input place or the variables (that are used to set the values to the output places), the transition operation (*Copy, Move, SendOverNetwork, Add, Sub, Prod, Div*), and the output place of the transitoin. The *Activations* objects must be added to the *GuardMapping* list as well. And when it is finished, the entire list should be added to the transition t_1 to store the first guard.

```
GuardMapping grdT11 = new GuardMapping();
grdT11.condition = T1Ct1;

grdT11.Activations.add(new Activation(t_1, "Halt", TransitionOperation. Move,
 "p_2"));
```

When all the gaurds are defined and added, the transition delay must be defined. Then the transition is added to the Petri net *Transitions* list.

```
t_1.Delay = 0;
pn.Transitions.add(t_1);
```

When the output channel is connected to a transition, The Activation object must contain the transition ID, the *transitionOperation* (*SendOverNetwork*), and the output channel ID of the same OER-TPN.

```
grdT3.Activations.add(new Activation(t_o2, "p_2",
TransitionOperation.SendOverNetwork, "p_o2"));
```

To run any Petri net and view the *GUI* for it as shown in figure 8.10

```
pn.Delay = 3000;
PetriNetWindow frame = new PetriNetWindow(false);
frame.petriNet = pn;
frame.setVisible(true);
```

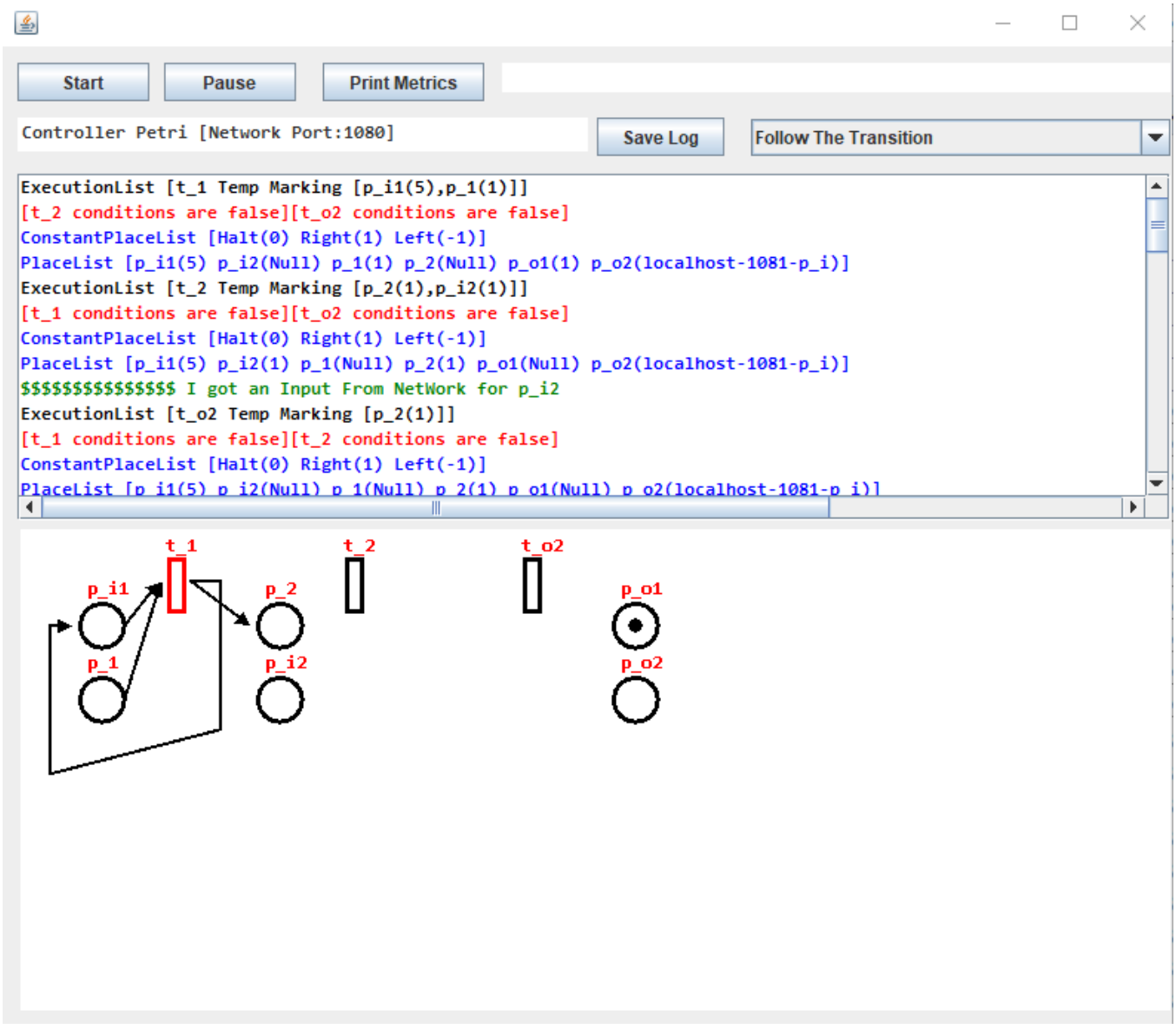


Figure 8.10 The controller GUI

Code sequence 1: The controller

`package OER_TPN_LAB;`
 	
`import Components.Activation;`
`import Components.Condition;`

```
import Components.GuardMapping;
import Components.PetriNet;
import Components.PetriNetWindow;
import Components.PetriTransition;
import DataObjects.DataInteger;
import DataObjects.DataTransfer;
import DataOnly.TransferOperation;
import Enumerations.LogicConnector;
import Enumerations.TransitionCondition;
import Enumerations.TransitionOperation;

public class Controller {
	public static void main(String[] args) {
		PetriNet pn = new PetriNet();
		pn.PetriNetName = "Controller Petri";
		pn.NetworkPort = 1080;

		DataInteger p_i1 = new DataInteger();
		p_i1.SetName("p_i1");
		pn.PlaceList.add(p_i1);

		DataInteger p_i2 = new DataInteger();
		p_i2.SetName("p_i2");
		pn.PlaceList.add(p_i2);

		DataInteger p_1 = new DataInteger();
		p_1.SetName("p_1");
		p_1.SetValue(0);
		pn.PlaceList.add(p_1);

		DataInteger p_2 = new DataInteger();
		p_2.SetName("p_2");
		pn.PlaceList.add(p_2);

		DataInteger p_o1 = new DataInteger();
		p_o1.SetName("p_o1");
		pn.PlaceList.add(p_o1);

		DataTransfer p_o2 = new DataTransfer();
		p_o2.SetName("p_o2");
		p_o2.Value = new TransferOperation("localhost", "1081", "p_i");
		pn.PlaceList.add(p_o2);

		DataInteger Halt = new DataInteger();
		Halt.SetName("Halt");
		Halt.SetValue(0);
		pn.ConstantPlaceList.add(Halt);

		DataInteger Right = new DataInteger();
		Right.SetName("Right");
		Right.SetValue(1);
		pn.ConstantPlaceList.add(Right);

		DataInteger Left = new DataInteger();
		Left.SetName("Left");
		Left.SetValue(-1);
		pn.ConstantPlaceList.add(Left);

		// ----------------------------------T_1-----------------------------------
-----
		PetriTransition t_1 = new PetriTransition(pn);
```

```
            t_1.TransitionName = "t_1";
            t_1.InputPlaceName.add("p_i1");
            t_1.InputPlaceName.add("p_1");

            // ---------grd1--------------------------
            Condition T1Ct1 = new Condition(t_1, "p_i1", TransitionCondition.NotNull);
            Condition T1Ct2 = new Condition(t_1, "p_1", TransitionCondition.NotNull);
            Condition T1Ct3 = new Condition(t_1, "p_i1", TransitionCondition.Equal,
"p_1");

            T1Ct2.SetNextCondition(LogicConnector.AND, T1Ct3);
            T1Ct1.SetNextCondition(LogicConnector.AND, T1Ct2);

            GuardMapping grdT11 = new GuardMapping();
            grdT11.condition = T1Ct1;

            grdT11.Activations.add(new Activation(t_1, "Halt", TransitionOperation.Move,
"p_2"));

            t_1.GuardMappingList.add(grdT11);

            // ---------grd2--------------------------------
            Condition T1Ct4 = new Condition(t_1, "p_i1", TransitionCondition.NotNull);
            Condition T1Ct5 = new Condition(t_1, "p_1", TransitionCondition.NotNull);
            Condition T1Ct6 = new Condition(t_1, "p_i1", TransitionCondition.MoreThan,
"p_1");

            T1Ct5.SetNextCondition(LogicConnector.AND, T1Ct6);
            T1Ct4.SetNextCondition(LogicConnector.AND, T1Ct5);

            GuardMapping grdT12 = new GuardMapping();
            grdT12.condition = T1Ct4;

            grdT12.Activations.add(new Activation(t_1, "Right",
TransitionOperation.Move, "p_2"));
            grdT12.Activations.add(new Activation(t_1, "p_i1", TransitionOperation.Move,
"p_i1"));
            t_1.GuardMappingList.add(grdT12);

            // -----------grd3-------------------------------------
            Condition T1Ct7 = new Condition(t_1, "p_i1", TransitionCondition.NotNull);
            Condition T1Ct8 = new Condition(t_1, "p_1", TransitionCondition.NotNull);
            Condition T1Ct9 = new Condition(t_1, "p_i1", TransitionCondition.LessThan,
"p_1");

            T1Ct8.SetNextCondition(LogicConnector.AND, T1Ct9);
            T1Ct7.SetNextCondition(LogicConnector.AND, T1Ct8);

            GuardMapping grdT13 = new GuardMapping();
            grdT13.condition = T1Ct7;

            grdT13.Activations.add(new Activation(t_1, "Left", TransitionOperation.Move,
"p_2"));
            grdT13.Activations.add(new Activation(t_1, "p_i1", TransitionOperation.Move,
"p_i1"));
            t_1.GuardMappingList.add(grdT13);

            t_1.Delay = 0;
            pn.Transitions.add(t_1);
```

```
            // ---------------------------------T_2-----------------------------------
-----
            PetriTransition t_2 = new PetriTransition(pn);
            t_2.TransitionName = "t_2";
            t_2.InputPlaceName.add("p_2");
            t_2.InputPlaceName.add("p_i2");

            Condition T2Ct1 = new Condition(t_2, "p_2", TransitionCondition.NotNull);
            Condition T2Ct2 = new Condition(t_2, "p_i2", TransitionCondition.NotNull);
            T2Ct1.SetNextCondition(LogicConnector.AND, T2Ct2);

            GuardMapping grdT2 = new GuardMapping();
            grdT2.condition = T2Ct1;

            grdT2.Activations.add(new Activation(t_2, "p_i2", TransitionOperation.Move,
"p_1"));
            grdT2.Activations.add(new Activation(t_2, "p_i2", TransitionOperation.Move,
"p_o1"));
            t_2.GuardMappingList.add(grdT2);

            t_2.Delay = 0;
            pn.Transitions.add(t_2);

            // --------------------T_o2-----------------------------------------
            PetriTransition t_o2 = new PetriTransition(pn);
            t_o2.TransitionName = "t_o2";
            t_o2.InputPlaceName.add("p_2");

            Condition T3Ct1 = new Condition(t_o2, "p_2", TransitionCondition.NotNull);
            GuardMapping grdT3 = new GuardMapping();
            grdT3.condition = T3Ct1;

            grdT3.Activations.add(new Activation(t_o2, "p_2",
TransitionOperation.SendOverNetwork, "p_o2"));
            grdT3.Activations.add(new Activation(t_o2, "p_2", TransitionOperation.Move,
"p_2"));
            t_o2.GuardMappingList.add(grdT3);

            t_o2.Delay = 0;
            pn.Transitions.add(t_o2);



            System.out.println("Exp1 started \n ------------------------------");
            pn.Delay = 3000;
            // pn.Start();

            PetriNetWindow frame = new PetriNetWindow(false);
            frame.petriNet = pn;
            frame.setVisible(true);

      }
}
```

Code sequence 2: The Robot

```
package OER_TPN_LAB;

import Components.Activation;
import Components.Condition;
```

```
import Components.GuardMapping;
import Components.PetriNet;
import Components.PetriNetWindow;
import Components.PetriTransition;
import DataObjects.DataInteger;
import DataObjects.DataTransfer;
import DataOnly.TransferOperation;
import Enumerations.LogicConnector;
import Enumerations.TransitionCondition;
import Enumerations.TransitionOperation;

public class Robot {
	public static void main(String[] args) {
		PetriNet pn = new PetriNet();
		pn.PetriNetName = "Robot Petri";
		pn.NetworkPort = 1081;

		DataInteger p_i = new DataInteger();
		p_i.SetName("p_i");
		pn.PlaceList.add(p_i);

		DataTransfer p_o = new DataTransfer();
		p_o.SetName("p_o");
		p_o.Value = new TransferOperation("localhost", "1080", "p_i2");
		pn.PlaceList.add(p_o);

		DataInteger p_0 = new DataInteger();
		p_0.SetName("p_0");
		p_0.SetValue(0);
		pn.PlaceList.add(p_0);

		DataInteger p_1 = new DataInteger();
		p_1.SetName("p_1");
		pn.PlaceList.add(p_1);

		// ----------------------------------T_1-----------------------------------
-----
		PetriTransition t_1 = new PetriTransition(pn);
		t_1.TransitionName = "t_1";
		t_1.InputPlaceName.add("p_i");
		t_1.InputPlaceName.add("p_0");

		Condition T1Ct1 = new Condition(t_1, "p_i", TransitionCondition.NotNull);
		Condition T1Ct2 = new Condition(t_1, "p_0", TransitionCondition.NotNull);
		T1Ct1.SetNextCondition(LogicConnector.AND, T1Ct2);

		GuardMapping grdT1 = new GuardMapping();
		grdT1.condition = T1Ct1;

		grdT1.Activations.add(new Activation(t_1, t_1.InputPlaceName,
TransitionOperation.Add, "p_1"));
		t_1.GuardMappingList.add(grdT1);

		t_1.Delay = 0;
		pn.Transitions.add(t_1);

		// ----------------------------------T_2-----------------------------------
-----
		PetriTransition t_2 = new PetriTransition(pn);
		t_2.TransitionName = "t_2";
		t_2.InputPlaceName.add("p_1");
```

```
            Condition T2Ct1 = new Condition(t_2, "p_1", TransitionCondition.NotNull);

            GuardMapping grdT2 = new GuardMapping();
            grdT2.condition = T2Ct1;

            grdT2.Activations.add(new Activation(t_2, "p_1",
TransitionOperation.SendOverNetwork, "p_o"));
            grdT2.Activations.add(new Activation(t_2, "p_1", TransitionOperation.Move,
"p_0"));
            t_2.GuardMappingList.add(grdT2);

            t_2.Delay = 2;
            pn.Transitions.add(t_2);

            System.out.println("Exp1 started \n ------------------------------");
            pn.Delay = 3000;
            // pn.Start();

            PetriNetWindow frame = new PetriNetWindow();
            frame.petriNet = pn;
            frame.setVisible(true);

      }
}
```

D. Multi-Tasking Implementation

This section presents a second implementation of the previous example (the Robot system). The implementation can be found in https://bitbucket.org/dahliajj/multi-tasking-implementation_galvanizing-line.git/src which is an eclipse workspace. To test the application, run the Main class only.

Figure 8.11 shows the class diagram for the entire system. For each place type, a unique class must be created (For example places of Integer type, Float types, String Types,…) and must contain Get() and Set() methods so they can be used by the transitions' guards and mappings. In the Robot example, we only need to use IntPlace place type, which contains an Integer attribute. The IntPlaceHandler has an ArrayList of IntPlace objects, the Robot class has its own ArrayList of IntPlaces, the Robot transitions access this ArrayList to execute their guards and mappings when they are fired. The same thing goes for the Controller class.

Each transition that is created must have its unique guards and mappings that are implemented in the GuardsMappings() method. The rest of the methods are common. Code sequence 5 shows an example of one transition (The controller t_o2 transition): This transition output place is the Robot input channel; that's why it is needed to access the Robot PlaceHandler. Each Transition class has two types of constructors: one with the fixed delay and the other for the delay duration ([eet, let]). The transition delay method is common for all the transition and it will be called in the Main class.

```
if (!PH.GetPlaceByName("p_2").IsNull()) {

      RobotPH.GetPlaceByName("p_i").Set((Integer)PH.GetPlaceByName("p_2").Get());
                  PH.GetPlaceByName("p_2").Set(null);
                  return true;
            }
            return false;
      }
```

The PlaceHandler provides an access by place name method, then the Get() and Set() methods can be used. The reason behind having a place handler to control access to the places.

Another example for a guard and mapping for the Controller transition t_1:

```
if (!PH.GetPlaceByName("p_i1").IsNull() && !PH.GetPlaceByName("p_1").IsNull()) {
                    if (PH.GetPlaceByName("p_i1").Get() == PH.GetPlaceByName("p_1").Get()) {
                          PH.GetPlaceByName("p_2").Set(0);
                          PH.GetPlaceByName("p_i1").Set(null);
                          PH.GetPlaceByName("p_1").Set(null);
                          return true;
                    }
                    if ((Integer) PH.GetPlaceByName("p_i1").Get() > (Integer)
PH.GetPlaceByName("p_1").Get()) {
                          PH.GetPlaceByName("p_2").Set(1);
                          PH.GetPlaceByName("p_1").Set(null);
                          return true;
                    }
                    if ((Integer) PH.GetPlaceByName("p_i1").Get() < (Integer)
PH.GetPlaceByName("p_1").Get()) {
                          PH.GetPlaceByName("p_2").Set(-1);
                          PH.GetPlaceByName("p_1").Set(null);
                          return true;
                    }
              }
              return false;
        }
```

Code sequences 3-7 contain the code highlights from the simpleOERTPN workspace.

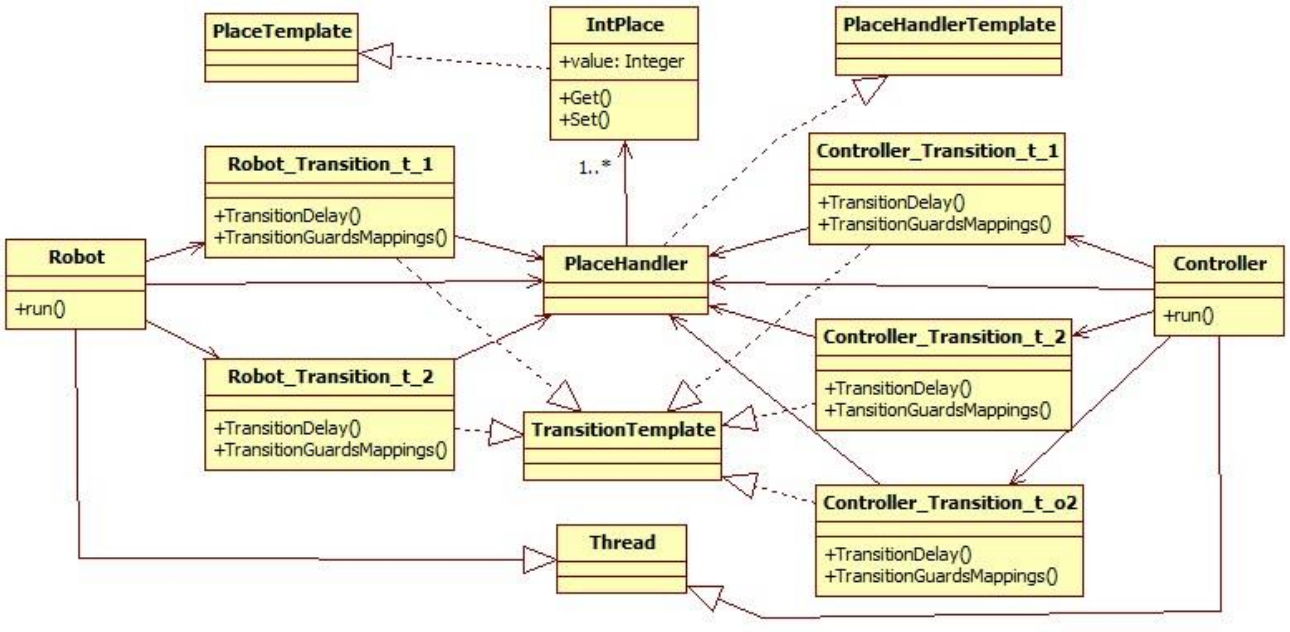


Figure 8.11 The class diagram for the Robot system

Code sequence 3: IntPlace class

`import Interfaces.PlaceTemplate;`
 	
`public class IntPlace implements PlaceTemplate {`

```
	Integer value;
	String Name;

	public IntPlace(String name, Object value) {
		this.Init(name, value);
	}

	@Override
	public Object Get() {
		return this.value;
	}

	@Override
	public void Set(Object value) {
		this.value = (Integer) value;
	}

	@Override
	public String GetPlaceName() {
		return this.Name;
	}

	@Override
	public void SetPlaceName(String name) {
		this.Name = name;
	}

	@Override
	public String Print() {
		return "[" + this.Name + "=" + this.value + "]";
	}

	@Override
	public void Init(String name, Object value) {
		this.SetPlaceName(name);
		this.Set(value);
	}

	@Override
	public Boolean IsNull() {
		return this.Get() == null;
	}
}
```

Code sequence 4: IntPlaceHandler class

```
import java.util.ArrayList;

import Interfaces.PlaceHandlerTemplate;
import Interfaces.PlaceTemplate;

public class IntPlaceHandler implements PlaceHandlerTemplate {

	ArrayList<PlaceTemplate> List;

	public IntPlaceHandler() {
		List = new ArrayList<PlaceTemplate>();
	}

	@Override
	public void AddPlace(PlaceTemplate place) {
```

```
                List.add(place);
        }

        @Override
        public PlaceTemplate GetPlaceByName(String Name) {
                for (PlaceTemplate placeTemplate : List) {
                        if (placeTemplate.GetPlaceName() == Name) {
                                return placeTemplate;
                        }
                }
                return null;
        }

        @Override
        public String PrintAllPlaces() {
                String toPrint = "";
                for (PlaceTemplate placeTemplate : List) {
                        toPrint = toPrint.concat(placeTemplate.Print());
                }
                return toPrint;
        }

}
```

Code sequence 5: An example of one Controller Transition t_o2

```
import Interfaces.PlaceHandlerTemplate;
import Interfaces.TransitionTemplate;

public class Controller_Transition_t_o2 implements TransitionTemplate {
        Integer timeUnitControl = 500;
        Integer eet;
        Integer let;
        String Name;
        PlaceHandlerTemplate PH;
        PlaceHandlerTemplate RobotPH;
        public Controller_Transition_t_o2(String name, PlaceHandlerTemplate PH, Integer
delay) {
                this.Init(name, PH);
                this.SetDelay(delay);
        }

        public Controller_Transition_t_o2(String name, PlaceHandlerTemplate PH, Integer
eet, Integer let) {
                this.Init(name, PH);
                this.SetDelayInRange(eet, let);
        }

        @Override
        public void TransitionDelay() {
                try {
                        if (this.let == null) {
                                Thread.sleep(this.eet * timeUnitControl);
                        } else {
                                Thread.sleep(Math.round(Math.random() * (this.let - this.eet) +
this.eet) * timeUnitControl);
                        }
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
```

```
	}

	@Override
	public Boolean TransitionGuardsMappings() {
		TransitionDelay();


		if (!PH.GetPlaceByName("p_2").IsNull()) {


	RobotPH.GetPlaceByName("p_i").Set((Integer)PH.GetPlaceByName("p_2").Get());
			PH.GetPlaceByName("p_2").Set(null);

			return true;
		}
		return false;
	}

	@Override
	public void Init(String name, PlaceHandlerTemplate PH) {
		this.PH = PH;
		this.Name = name;
	}

	@Override
	public void SetDelay(int value) {
		this.eet = value;
	}

	@Override
	public void SetDelayInRange(int eet, int let) {
		this.eet = eet;
		this.let = let;
	}
}
```

Code sequence 6: Controller class

```
import java.util.Scanner;

import Interfaces.PlaceTemplate;

public class Controller extends Thread {

	boolean stop = false;
	public Robot r;
	IntPlaceHandler PH = new IntPlaceHandler();

	Controller_Transition_t_1 t_1;
	Controller_Transition_t_2 t_2;
	Controller_Transition_t_o2 t_o2;
	Scanner in = new Scanner(System.in);

	public void run() {

		PH.AddPlace(new IntPlace("p_i1", null));
		PH.AddPlace(new IntPlace("p_1", 0));
		PH.AddPlace(new IntPlace("p_o1", null));
		PH.AddPlace(new IntPlace("p_2", null));
		PH.AddPlace(new IntPlace("p_i2", null));
```

```
		t_1 = new Controller_Transition_t_1("t_1", PH, 0);
		t_2 = new Controller_Transition_t_2("t_2", PH, 0);
		t_o2 = new Controller_Transition_t_o2("t_o2", PH, 0);
		t_o2.RobotPH = r.PH;// this transition has an output channel connected to
the robot

		System.out.println("Controller: Input p_i1 value");
		this.PH.GetPlaceByName("p_i1").Set(Integer.parseInt(in.nextLine()));

		while (!stop) {

			t_1.TransitionGuardsMappings();

			t_2.TransitionGuardsMappings();

			t_o2.TransitionGuardsMappings();

			// For slower printing on the console
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}

	}

	public void StopThread() {
		this.stop = true;
	}

}
```

Code sequence 5: An example of one Robot Transition t_2

```
import Interfaces.PlaceHandlerTemplate;
import Interfaces.TransitionTemplate;

public class Robot_Transition_t_2 implements TransitionTemplate {
	Integer timeUnitControl = 500;
	Integer eet;
	Integer let;
	String Name;
	PlaceHandlerTemplate PH;
	PlaceHandlerTemplate ControllerPH;

	public Robot_Transition_t_2(String name, PlaceHandlerTemplate PH, Integer delay) {
		this.Init(name, PH);
		this.SetDelay(delay);
	}

	public Robot_Transition_t_2(String name, PlaceHandlerTemplate PH, Integer eet,
Integer let) {
		this.Init(name, PH);
		this.SetDelayInRange(eet, let);
	}

	@Override
	public void TransitionDelay() {
		try {
```

```
                    if (this.let == null) {
                          Thread.sleep(this.eet * timeUnitControl);
                    } else {
                          Thread.sleep(Math.round(Math.random() * (this.let - this.eet) +
this.eet) * timeUnitControl);
                    }
             } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
             }
      }

      @Override
      public Boolean TransitionGuardsMappings() {
             TransitionDelay();
             String toPrint="--------------Robot-------------------\n";

             if (!PH.GetPlaceByName("p_1").IsNull()) {

                    toPrint=toPrint.concat(Print()+"\n");

      toPrint=toPrint.concat("ControllerPH\n"+ControllerPH.PrintAllPlaces()+"\n");

                    PH.GetPlaceByName("p_o").Set(PH.GetPlaceByName("p_1").Get());
                    PH.GetPlaceByName("p_0").Set(PH.GetPlaceByName("p_1").Get());

      ControllerPH.GetPlaceByName("p_i2").Set(PH.GetPlaceByName("p_1").Get());
                    PH.GetPlaceByName("p_1").Set(null);

                    return true;
             }
             return false;
      }

      @Override
      public void Init(String name, PlaceHandlerTemplate PH) {
             this.PH = PH;
             this.Name = name;
      }

      @Override
      public void SetDelay(int value) {
             this.eet = value;
      }

      @Override
      public void SetDelayInRange(int eet, int let) {
             this.eet = eet;
             this.let = let;
      }


}
```

Code sequence 6: Robot class

```
public class Robot extends Thread {

      boolean stop = false;
```

```
	IntPlaceHandler PH = new IntPlaceHandler();

	public Controller c;

	Robot_Transition_t_1 t_1;
	Robot_Transition_t_2 t_2;

	public void run() {

		PH.AddPlace(new IntPlace("p_0", 0));
		PH.AddPlace(new IntPlace("p_1", null));
		PH.AddPlace(new IntPlace("p_i", null));
		PH.AddPlace(new IntPlace("p_o", null));
		t_1 = new Robot_Transition_t_1("t_1", PH, 0);
		t_2 = new Robot_Transition_t_2("t_2", PH, 5);
		t_2.ControllerPH = c.PH;// this transition has an output channel connected
to the controller

		while (!stop) {
			t_1.TransitionGuardsMappings();

			t_2.TransitionGuardsMappings();

			// For slower printing on the console
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}

	}

	public void StopThread() {
		this.stop = true;
	}

}
```

Code sequence 7: Main class

```
public class Main {

	public static void main(String[] args) {
		Controller c = new Controller();
		Robot r = new Robot ();
		c.r = r;
		r.c = c;
		c.start();
		r.start();
	}
}
```

E. Exercise 1:

Build a galvanizing line control application represented by the OER-TPN model previously presented in section 2.4. The control application receives from the user a technology described by a sequence of the form:

$$\sigma = R_0[0,0]*R_i[eet_i, let_i]*R_k[eet_k, let_k]*R_j[eet_j, let_j]* \ldots *R_n[0,0]$$

where R_k is the resource where the container is to be placed, eet_k represents the minimum time it must stay in R_k, and let_k represents the maximum time until which the container must be moved to the next resource.

The Architecture is comped of *Supervisor* and *Controller*.

Supervisor transforms the sequence $\sigma = R_0[0,0]*R_i[eet_i, let_i]* R_k[eet_k, let_k]* R_j[eet_j, let_j]* \cdots *R_n[0,0]$ in demands that are sent periodically and have to be executed by *Controller*.

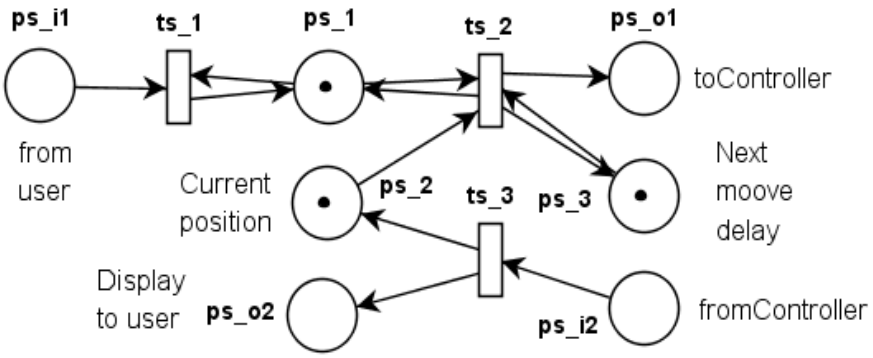


Figure 8.12 The Supervisor

The place types are as follows:

type(ps_i1) = Cl_ps-i1 {int r; int e, int l}
type(ps_1) = Cl_ps-1 extends List <ps_i1>;
type(ps_2) = type(ps_o1)=type(ps_3)=type(ps_i2)=type(ps_o2)= int;
init(): {ps_2.v=0; ps_3.delay=0; ps_o1=φ; ps_1.list }

The transitions guards and mappings are as follows:

t_s1:

ps_i1≠ null ➔ map(ps_1) : ps_1. addElement (ps_i1)

ts_2 [ps_3]:

(ps_1≠ null (has elements)) AND (ps_2≠ null) AND (ps_1.r ≠ ps_2)

➔ map(ps_o1): ps_o1 = ps_1.r //The resource r

map(ps_3): ps_3 = ps_1. e //The delay for ts_2

ts_3:

ps_i2≠ null ➔ map(ps_2): ps_2 = ps_i2

map(ps_o2): ps_o2 = ps_i2

Requirements:

1. Implement the Supervisor OER-TPN.
2. Connect the output channel p_o1 of the previously implemented Controller to the Supervisor input channel ps_i2.
3. The source code for the implementation is required (Present the whole workspace).

   Note:

   a. If you choose to implement with the OER-TPN Framework, you can use the following framework existing components for the supervisor implementation:
      - The data type: DataREL for place ps_i1
      - The data type: DataRELQueue for place ps_1
      - Set DataRELQueue: ps_1.Value.Size = 10
      - ts_1: Activation: AddElement
      - ts_2: create output place list that has the sequence (ps_o1, ps_3)
      - ts_2: Condition: HaveREL to check if ps_1 has items
      - ts_2: Activation PopElement_R_E to the output places list //this method also sets the delay for the transition with ps_3.
      - ts_2: Activation: ps_o1: SendROverNetwork //this method also checks if the second item from the InputPlaceName list(ps_2) != r so there is no need to write this condition.
      - Use GUI> InputREL to send data to the Supervisor

   b. If you choose to implement with Mulit-Tasking Implementation, make sure to create the new data types, new transitions and linking the Supervisor and Controller channels togethers.
4. Draw the state machine model for the galvanizing line system.

4. Example of an Application 2: Nets within nets (Parent and Child creation)

The newly created task is in its turn a subgraph representing an OETPN model. The OETPN that creates a new OETPN is called *parent* and the latest *child*. Recursively, the child can create other children. The parents and children are executed concurrently and independently. Fig. 8.12 describes an OETPN denoted PN1 that is the parent of the child OETPN PN2. The parent and the child can communicate through the token set in *p2*. The parent sends information to the child by the parameters of the active object set in *p2*. More than this, the parent creates the structure and the behavior of the PN2 when map1 is executed. When the execution of the task corresponding to PN2 ends, its marking becomes accessible to PN1 containing the information that the child leaves it.

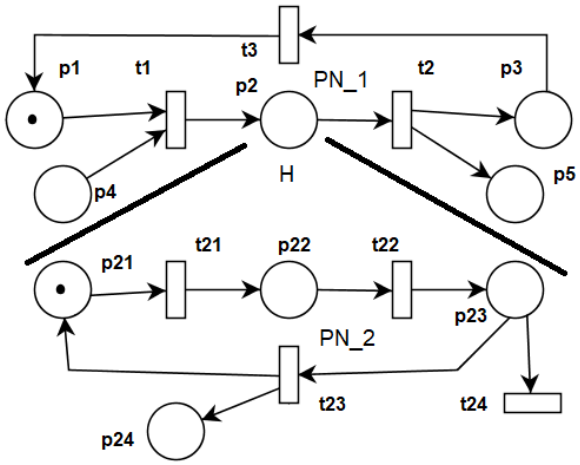


Fig. 8.12 Example of a hierarchical OETPN.

Solution:

A) The OER_TPN Model

The parent places have the types:

- type(p_1) = type(p_3) = type(p_4) = type(p_5) = float; the tokens are denoted by x_1, x_3, x_4 and x_5 respectively.
- type(p_2) = OETPN; the token is denoted by PN_2

The child places have the types:

- type(p_{21}) = type(p_{22}) = type(p_{23}) = type(p_{24}) = float; the tokens are denoted by x_{21}, x_{22}, x_{23} and x_{24} respectively.

The parent guards and mappings are:

- grd_1^1= ($x_1 \neq \varphi$) AND ($x_4 \leq 1$); map_1^1 instantiates the child $M(p_2) = PN_2$ being an active object with the marking $M^2 = [x_4, 0, 0, 0]$ (i.e. $M(p_{21}) = M(p_4)$ and the running state true).
- grd_1^2= ($x_1 \neq \varphi$) AND ($x_4 > 1$); map_1^2 instantiates the child $M(p_2) = PN_2$ being a passive object with the marking $M^2 = [x_4, 0, 0, 0]$ (i.e. $M(p_{21}) = M(p_4)$ and running state *false*.
- Transition t2 can be triggered only if PN2 finishes its execution (Passive) and can access its marking, for the example the grd and map goes as follows:
  grd_2= (PN2.state = Passive); map_2 $M(p_3)$= M(p24); M(p4)= M(p24);
- grd_3= (x3≠ φ); map_2 $M(p_1)$= M(p3);

The child guards and mappings are:

- grd_{21} = true; map_{21}: $x_{22} = x_{21}$;
- grd_{22} = true; map_{22}: $x_{23} = x_{22} + 0.1$;
- $grd_{23} = x_{23} < 2$; map_{23}: $x_{21} = x_{23}$;
- $grd_{24} = x_{23} \geq 2$; **StopPetriNet()**

B. Implementation with OER-TPN Framework

The implementation is given with the linke below:

https://bitbucket.org/dahliajj/oetpn_oertpn_framework/src/master/

It is an eclipse workspace, in the *OER_TPN_Lab* package. Make sure to navigate to one (NewOETPN folder). The implementation is found in the OETPN package> Exp1.

Code explanation:

In code sequence 6, we first create the subPetri as the design.

Transition t24 is the one responsible for stopping the thread of execution, see its mapping:

```
grdT24.Activations.add(new Activation(t24,"", TransitionOperation.StopPetriNet, ""));
```

After creating the subPetri, we need to create a constant value object of it so we can use if for t1:

```
DataSubPetriNet subPetriNet = new DataSubPetriNet();
            subPetriNet.SetName("SubPetri");
            SubPetri sptr= new SubPetri(spn);
            subPetriNet.SetValue(sptr);
            pn.ConstantPlaceList.add(subPetriNet);
```

Transition 1 first subguard:

1. To move the constant subPetri value to p2:

```
grdT1.Activations.add(new Activation(t1, "SubPetri", TransitionOperation.Move, "p2"));
```

2. To move the token from p4 to p2 and then to the subPetri place p21:

```
grdT1.Activations.add(new Activation(t1, "p4", TransitionOperation.Move, "p2-p21"));
```

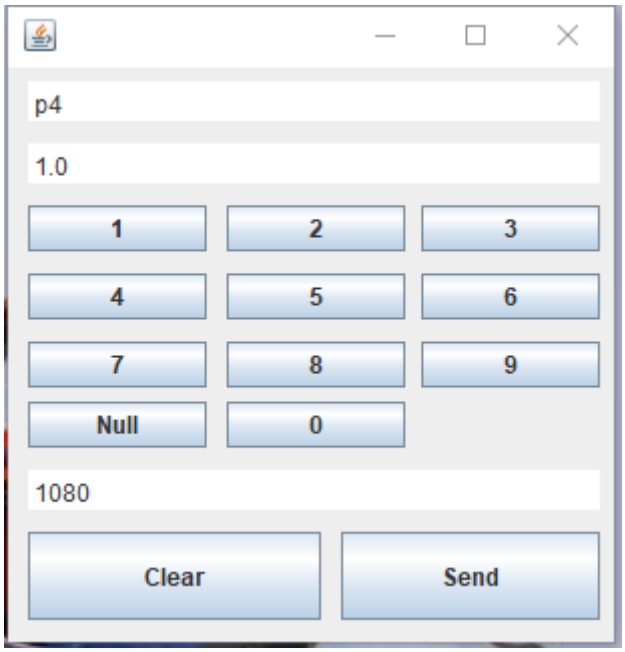
3. Activate the subPetri (start the thread of the child):

```
grdT1.Activations.add(new Activation(t1, "p2", transitionOperation.ActivateSubPetri, ""));
```

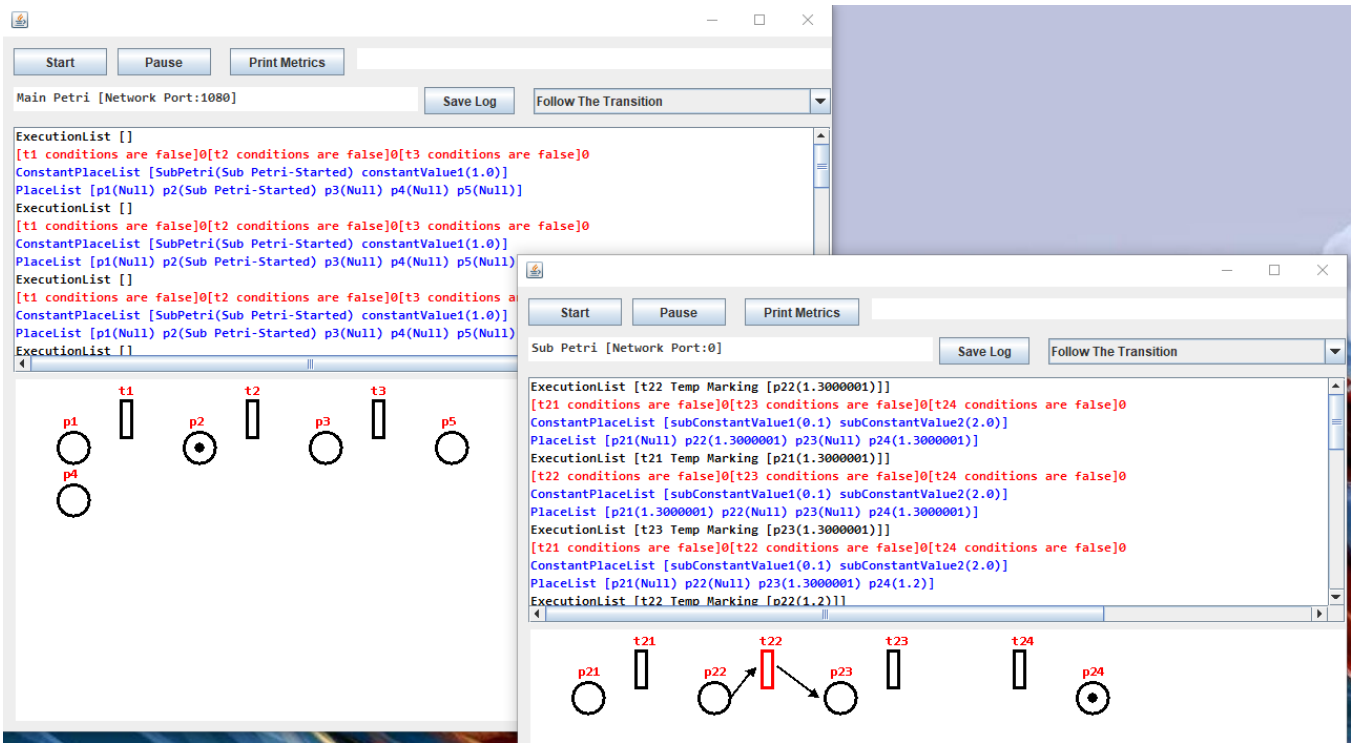
Transition 2 will have to check if the subPetri has finished its execution (if it becomes Passive) and it will also trigger if the subPetri is not started i.e. it is passive:

```
Condition T2Ct2 = new Condition(t2, "p2", TransitionCondition.SubPetriStopped);
```

Note: For running the code, you need to run the InputFload from the GUI package, and send 1 to p4 to PN1 as shown in figure 8.13. and run Exp1 and the subPetri will run automatically as shown in figure 8.14.



8.13 InputFload



8.14 Main Petri and SubPetri

Code sequence 6: Exp1

```
package OETPN;

import java.util.ArrayList;
import Components.Activation;
import Components.Condition;
```

```
import Components.GuardMapping;
import Components.PetriNet;
import Components.PetriNetWindow;
import Components.PetriTransition;
import DataObjects.DataFloat;
import DataObjects.DataSubPetriNet;
import DataOnly.SubPetri;
import Enumerations.LogicConnector;
import Enumerations.TransitionCondition;
import Enumerations.TransitionOperation;

public class Exp1 {

	public static void main(String[] args) {
		PetriNet pn = new PetriNet();
		pn.PetriNetName = "Main Petri";
		pn.NetworkPort = 1080;

		// ---------------------- sub petri ----------------------------------
		PetriNet spn = new PetriNet();
		spn.PetriNetName = "Sub Petri";

		DataFloat subConstantValue1 = new DataFloat();
		subConstantValue1.SetName("subConstantValue1");
		subConstantValue1.SetValue(0.1f);
		spn.ConstantPlaceList.add(subConstantValue1);

		DataFloat subConstantValue2 = new DataFloat();
		subConstantValue2.SetName("subConstantValue2");
		subConstantValue2.SetValue(2.0f);
		spn.ConstantPlaceList.add(subConstantValue2);

		DataFloat p21 = new DataFloat();
		p21.SetName("p21");
		spn.PlaceList.add(p21);

		DataFloat p22 = new DataFloat();
		p22.SetName("p22");
		spn.PlaceList.add(p22);

		DataFloat p23 = new DataFloat();
		p23.SetName("p23");
		spn.PlaceList.add(p23);

		DataFloat p24 = new DataFloat();
		p24.SetName("p24");
		//p24.SetValue(1.0f);
		spn.PlaceList.add(p24);

		// T21 ------------------------------------------------
		PetriTransition t21 = new PetriTransition(spn);
		t21.TransitionName = "t21";
		t21.InputPlaceName.add("p21");

		Condition T21Ct1 = new Condition(t21, "p21", TransitionCondition.NotNull);

		GuardMapping grdT21 = new GuardMapping();
		grdT21.condition = T21Ct1;
		grdT21.Activations.add(new Activation(t21, "p21", TransitionOperation.Move,
"p22"));
```

```
            t21.GuardMappingList.add(grdT21);
            t21.Delay = 0;
            spn.Transitions.add(t21);

            // T22 ------------------------------------------------
            PetriTransition t22 = new PetriTransition(spn);
            t22.TransitionName = "t22";
            t22.InputPlaceName.add("p22");

            Condition T22Ct1 = new Condition(t22, "p22", TransitionCondition.NotNull);

            GuardMapping grdT22 = new GuardMapping();
            grdT22.condition = T22Ct1;

            ArrayList<String> lstInput = new ArrayList<String>();
            lstInput.add("p22");
            lstInput.add("subConstantValue1");
            grdT22.Activations.add(new Activation(t22, lstInput,
TransitionOperation.Add, "p23"));

            t22.GuardMappingList.add(grdT22);
            t22.Delay = 0;
            spn.Transitions.add(t22);

            // T23 ------------------------------------------------
            PetriTransition t23 = new PetriTransition(spn);
            t23.TransitionName = "t23";
            t23.InputPlaceName.add("p23");

            Condition T23Ct1 = new Condition(t23, "p23", TransitionCondition.NotNull);
            Condition T23Ct2 = new Condition(t23, "p23", TransitionCondition.LessThan,
"subConstantValue2");
            T23Ct1.SetNextCondition(LogicConnector.AND, T23Ct2);

            GuardMapping grdT23 = new GuardMapping();
            grdT23.condition = T23Ct1;

            grdT23.Activations.add(new Activation(t23,"p23", TransitionOperation.Move,
"p24"));
            grdT23.Activations.add(new Activation(t23,"p23", TransitionOperation.Move,
"p21"));


            t23.GuardMappingList.add(grdT23);
            t23.Delay = 0;
            spn.Transitions.add(t23);


            // T24 ------------------------------------------------
            PetriTransition t24 = new PetriTransition(spn);
            t24.TransitionName = "t24";
            t24.InputPlaceName.add("p23");

            Condition T24Ct1 = new Condition(t24, "p23", TransitionCondition.NotNull);
            Condition T24Ct2 = new Condition(t24, "p23",
TransitionCondition.MoreThanOrEqual, "subConstantValue2");
            T24Ct1.SetNextCondition(LogicConnector.AND, T24Ct2);

            GuardMapping grdT24 = new GuardMapping();
            grdT24.condition = T24Ct1;
```

```
            grdT24.Activations.add(new Activation(t24,"",
TransitionOperation.StopPetriNet, ""));

            t24.GuardMappingList.add(grdT24);
            t24.Delay = 0;
            spn.Transitions.add(t24);

            spn.Delay = 3000;

            // ---------------------------------------------------------------------
            DataSubPetriNet subPetriNet = new DataSubPetriNet();
            subPetriNet.SetName("SubPetri");
            SubPetri sptr= new SubPetri(spn);
            subPetriNet.SetValue(sptr);
            pn.ConstantPlaceList.add(subPetriNet);

            DataFloat constantValue1 = new DataFloat();
            constantValue1.SetName("constantValue1");
            constantValue1.SetValue(1.0f);
            pn.ConstantPlaceList.add(constantValue1);

            DataFloat p1 = new DataFloat();
            p1.SetName("p1");
            p1.SetValue(1.0f);
            pn.PlaceList.add(p1);

            DataSubPetriNet p2 = new DataSubPetriNet();
            p2.SetName("p2");
            pn.PlaceList.add(p2);

            DataFloat p3 = new DataFloat();
            p3.SetName("p3");
            pn.PlaceList.add(p3);

            DataFloat p4 = new DataFloat();
            p4.SetName("p4");
            //p4.SetValue(1.0f);//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<test
            pn.PlaceList.add(p4);

            DataFloat p5 = new DataFloat();
            p5.SetName("p5");
            pn.PlaceList.add(p5);

            // T1 ------------------------------------------------
            PetriTransition t1 = new PetriTransition(pn);
            t1.TransitionName = "t1";
            t1.InputPlaceName.add("p1");
            t1.InputPlaceName.add("p4");

            Condition T1Ct1 = new Condition(t1, "p1", TransitionCondition.NotNull);
            Condition T1Ct2 = new Condition(t1, "p4",
TransitionCondition.LessThanOrEqual, "constantValue1");
            T1Ct1.SetNextCondition(LogicConnector.AND, T1Ct2);

            GuardMapping grdT1 = new GuardMapping();
            grdT1.condition = T1Ct1;
            grdT1.Activations.add(new Activation(t1, "SubPetri",
TransitionOperation.Move, "p2"));
            grdT1.Activations.add(new Activation(t1, "p4", TransitionOperation.Move,
"p2-p21"));
```

```
            grdT1.Activations.add(new Activation(t1, "p2",
TransitionOperation.ActivateSubPetri, ""));

            t1.GuardMappingList.add(grdT1);

            Condition T1Ct3 = new Condition(t1, "p1", TransitionCondition.NotNull);
            Condition T1Ct4 = new Condition(t1, "p4", TransitionCondition.MoreThan,
"constantValue1");
            T1Ct3.SetNextCondition(LogicConnector.AND, T1Ct4);

            GuardMapping grd2T1 = new GuardMapping();
            grd2T1.condition = T1Ct3;
            grd2T1.Activations.add(new Activation(t1, "SubPetri",
TransitionOperation.Copy, "p2"));
            grd2T1.Activations.add(new Activation(t1, "p4", TransitionOperation.Move,
"p2-p21"));

            t1.GuardMappingList.add(grd2T1);

            t1.Delay = 0;
            pn.Transitions.add(t1);

            // T2 ------------------------------------------------
            PetriTransition t2 = new PetriTransition(pn);
            t2.TransitionName = "t2";
            t2.InputPlaceName.add("p2");

            Condition T2Ct1 = new Condition(t2, "p2", TransitionCondition.NotNull);
            Condition T2Ct2 = new Condition(t2, "p2",
TransitionCondition.SubPetriStopped);
            T2Ct1.SetNextCondition(LogicConnector.AND, T2Ct2);

            GuardMapping grdT2 = new GuardMapping();
            grdT2.condition = T2Ct1;
            grdT2.Activations.add(new Activation(t2, "p2-p24", TransitionOperation.Copy,
"p3"));
            grdT2.Activations.add(new Activation(t2, "p2-p24", TransitionOperation.Move,
"p5"));

            t2.GuardMappingList.add(grdT2);
            t2.Delay = 0;
            pn.Transitions.add(t2);

            // T3 ------------------------------------------------
            PetriTransition t3 = new PetriTransition(pn);
            t3.TransitionName = "t3";
            t3.InputPlaceName.add("p3");

            Condition T3Ct1 = new Condition(t3, "p3", TransitionCondition.NotNull);

            GuardMapping grdT3 = new GuardMapping();
            grdT3.condition = T3Ct1;
            grdT3.Activations.add(new Activation(t3, "p3", TransitionOperation.Move,
"p1"));

            t3.GuardMappingList.add(grdT3);
            t3.Delay = 0;
            pn.Transitions.add(t3);

            System.out.println("Exp1 started \n ------------------------------");
            pn.Delay = 3000;
```

```
            PetriNetWindow frame = new PetriNetWindow(false);
            frame.petriNet = pn;
            frame.setVisible(true);
      }
}
```

C. Multi-Tasking Implementation

The implementation is similar to the Galvanizing line mulit-tasking implementation, the difference is that two new place types are added: FloatPlace and Child (which is an OER-TPN place type). Transition t_2 of the parent checks the runningState Boolean attribute from the Child, this transition can only be triggered if the runningState is false so it can have access to the Child's marking and moves the token from p_21 of the Child to its output places p_3 and P_5. The implementation can be found in https://bitbucket.org/dahliajj/multi-tasking-implementation_parentchild/src/main/ which is an eclipse workspace.

Code sequence 7: FloatPlace

```
import Interfaces.PlaceTemplate;

public class FloatPlace implements PlaceTemplate {
      Float value;
      String Name;

      public FloatPlace(String name, Object value) {
            this.Init(name, value);
      }

      @Override
      public Object Get() {
            return this.value;
      }

      @Override
      public void Set(Object value) {
            this.value = (Float) value;
      }

      @Override
      public String GetPlaceName() {
            return this.Name;
      }

      @Override
      public void SetPlaceName(String name) {
            this.Name = name;
      }

      @Override
```

```
	public String Print() {
		return "[" + this.Name + "=" + this.value + "]";
	}

	@Override
	public void Init(String name, Object value) {
		this.SetPlaceName(name);
		this.Set(value);
	}

	@Override
	public Boolean IsNull() {
		return this.Get() == null;
	}
}
```

Code sequence 8: Child place (Active Token)

```
import Interfaces.PlaceTemplate;

public class Child extends Thread implements PlaceTemplate {
	String name;
	boolean stop = false;
	PlaceHandler PH = new PlaceHandler();

	public Parent c;

	Child_Transition_t_1 t_1;
	Child_Transition_t_2 t_2;
	Child_Transition_t_3 t_3;
	Child_Transition_t_4 t_4;

	Boolean runningState = false;

	public Child(String name) {
		PH.AddPlace(new FloatPlace("p_21", null));
		PH.AddPlace(new FloatPlace("p_22", null));
		PH.AddPlace(new FloatPlace("p_23", null));
		PH.AddPlace(new FloatPlace("p_24", null));
		t_1 = new Child_Transition_t_1("t_1", PH, 0);
		t_2 = new Child_Transition_t_2("t_2", PH, 0);
		t_3 = new Child_Transition_t_3("t_3", PH, 0);
		t_4 = new Child_Transition_t_4("t_4", PH, 0);
		t_4.c = this;
		this.SetPlaceName(name);
	}

	public void run() {
		runningState = true;
```

```
		while (!stop) {
			t_1.TransitionGuardsMappings();
			t_2.TransitionGuardsMappings();
			t_3.TransitionGuardsMappings();
			t_4.TransitionGuardsMappings();

			// For slower printing on the console
			try {
				Thread.sleep(1000);
			} catch (InterruptedException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
		runningState = false;
	}

	public void StopThread() {
		this.stop = true;
	}

	@Override
	public Object Get() {
		return this;
	}

	public boolean GetRunningState() {
		return this.runningState;
	}

	@Override
	public void Set(Object value) {
		//
	}

	@Override
	public String GetPlaceName() {
		return this.name;
	}

	@Override
	public void SetPlaceName(String name) {
		this.name = name;
	}

	@Override
	public String Print() {
		return "[" + this.name + "=" + this.runningState + "]";
	}
```

```
	@Override
	public void Init(String name, Object value) {
		this.SetPlaceName(name);
		this.Set(value);
	}

	@Override
	public Boolean IsNull() {
		return this.Get() == null;
	}
}
```

Code sequence 9: Parent transition t_1

```
import Interfaces.PlaceHandlerTemplate;
import Interfaces.TransitionTemplate;

public class Parent_Transition_t_1 implements TransitionTemplate {
	Integer timeUnitControl = 500;
	Integer eet;
	Integer let;
	String Name;
	PlaceHandlerTemplate PH;
	PlaceHandlerTemplate ChildPH;

	public Parent_Transition_t_1(String name, PlaceHandlerTemplate PH, Integer delay) {
		this.Init(name, PH);
		this.SetDelay(delay);
	}

	public Parent_Transition_t_1(String name, PlaceHandlerTemplate PH, Integer eet, Integer let)
{
		this.Init(name, PH);
		this.SetDelayInRange(eet, let);
	}

	@Override
	public void TransitionDelay() {
		try {
			if (this.let == null) {
				Thread.sleep(this.eet * timeUnitControl);
			} else {
				Thread.sleep(Math.round(Math.random() * (this.let - this.eet) + this.eet)
* timeUnitControl);
			}
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
```

```
		}
	}

	@Override
	public Boolean TransitionGuardsMappings() {
		TransitionDelay();
		String toPrint = "--------------Parent--------------\n";

		if (!PH.GetPlaceByName("p_1").IsNull() && !PH.GetPlaceByName("p_4").IsNull())
{
			if ((Float) (PH.GetPlaceByName("p_4").Get()) <= 1.0f) {
				toPrint = toPrint.concat(Print() + "\n");

	ChildPH.GetPlaceByName("p_21").Set(PH.GetPlaceByName("p_4").Get());
				((Child) (PH.GetPlaceByName("p_2"))).start();

				PH.GetPlaceByName("p_1").Set(null);
				PH.GetPlaceByName("p_4").Set(null);

				toPrint = toPrint.concat(Print() + "\n");
				toPrint = toPrint.concat("-------------------------------------\n");

				System.out.println(toPrint);
				return true;
			}

			if (!PH.GetPlaceByName("p_1").IsNull() &&
!PH.GetPlaceByName("p_4").IsNull()) {
				if ((Float) (PH.GetPlaceByName("p_4").Get()) > 1.0f) {
					toPrint = toPrint.concat(Print() + "\n");

	ChildPH.GetPlaceByName("p_21").Set(PH.GetPlaceByName("p_4").Get());
					PH.GetPlaceByName("p_1").Set(null);
					PH.GetPlaceByName("p_4").Set(null);

					toPrint = toPrint.concat(Print() + "\n");
					toPrint = toPrint.concat("-------------------------------------\n");
					System.out.println(toPrint);
					return true;
				}
			}
		}return false;
	}

	@Override
	public void Init(String name, PlaceHandlerTemplate PH) {
		this.PH = PH;
		this.Name = name;
```

```
	}

	@Override
	public void SetDelay(int value) {
		this.eet = value;
	}

	@Override
	public void SetDelayInRange(int eet, int let) {
		this.eet = eet;
		this.let = let;
	}

	@Override
	public String Print() {
		return this.Name + "\n" + this.PH.PrintAllPlaces();
	}

}
```

Code sequence 10: Parent t_2 transition

```
import Interfaces.PlaceHandlerTemplate;
import Interfaces.TransitionTemplate;

public class Parent_Transition_t_2 implements TransitionTemplate {
	Integer timeUnitControl = 500;
	Integer eet;
	Integer let;
	String Name;
	PlaceHandlerTemplate PH;
	PlaceHandlerTemplate ChildPH;
	public Parent_Transition_t_2(String name, PlaceHandlerTemplate PH, Integer delay) {
		this.Init(name, PH);
		this.SetDelay(delay);
	}

	public Parent_Transition_t_2(String name, PlaceHandlerTemplate PH, Integer eet, Integer let)
{
		this.Init(name, PH);
		this.SetDelayInRange(eet, let);
	}

	@Override
	public void TransitionDelay() {
		try {
			if (this.let == null) {
				Thread.sleep(this.eet * timeUnitControl);
```

```
                        } else {
                                Thread.sleep(Math.round(Math.random() * (this.let - this.eet) + this.eet)
* timeUnitControl);
                        }
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        @Override
        public Boolean TransitionGuardsMappings() {
                TransitionDelay();
                String toPrint="----------Parent----------\n";
                if ( !PH.GetPlaceByName("p_2").IsNull()&&
(((Child)(PH.GetPlaceByName("p_2"))).GetRunningState())==false) {
                        toPrint = toPrint.concat(Print() + "\n");

                        PH.GetPlaceByName("p_3").Set(ChildPH.GetPlaceByName("p_21").Get());
                        PH.GetPlaceByName("p_5").Set(ChildPH.GetPlaceByName("p_21").Get());
                        PH.GetPlaceByName("p_2").Set(null);

                        toPrint = toPrint.concat(Print() + "\n");
                        toPrint = toPrint.concat("-----------------------------------\n");
                        System.out.println(toPrint);
                        return true;
                }
                return false;
        }

        @Override
        public void Init(String name, PlaceHandlerTemplate PH) {
                this.PH = PH;
                this.Name = name;
        }

        @Override
        public void SetDelay(int value) {
                this.eet = value;
        }

        @Override
        public void SetDelayInRange(int eet, int let) {
                this.eet = eet;
                this.let = let;
        }

        @Override
```

```
	public String Print() {
		return this.Name + "\n" + this.PH.PrintAllPlaces();
	}
}
```

Code sequence 11: Child t_1 transition

```
import Interfaces.PlaceHandlerTemplate;
import Interfaces.TransitionTemplate;

public class Child_Transition_t_1 implements TransitionTemplate {
	Integer timeUnitControl = 500;
	Integer eet;
	Integer let;
	String Name;
	PlaceHandlerTemplate PH;

	public Child_Transition_t_1(String name, PlaceHandlerTemplate PH, Integer delay) {
		this.Init(name, PH);
		this.SetDelay(delay);
	}

	public Child_Transition_t_1(String name, PlaceHandlerTemplate PH, Integer eet, Integer let)
{
		this.Init(name, PH);
		this.SetDelayInRange(eet, let);
	}

	@Override
	public void TransitionDelay() {
		try {
			if (this.let == null) {
				Thread.sleep(this.eet * timeUnitControl);
			} else {
				Thread.sleep(Math.round(Math.random() * (this.let - this.eet) + this.eet)
* timeUnitControl);
			}
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	@Override
	public Boolean TransitionGuardsMappings() {
		TransitionDelay();
		String toPrint="----------Child----------\n";
```

```
                if (!PH.GetPlaceByName("p_21").IsNull()) {
                        toPrint = toPrint.concat(Print() + "\n");

                        PH.GetPlaceByName("p_22").Set((Float)
PH.GetPlaceByName("p_21").Get());
                        PH.GetPlaceByName("p_21").Set(null);


                        toPrint = toPrint.concat(Print() + "\n");
                        toPrint = toPrint.concat("---------------------------------------\n");

                        System.out.println(toPrint);
                        return true;
                }
                return false;
        }

        @Override
        public void Init(String name, PlaceHandlerTemplate PH) {
                this.PH = PH;
                this.Name = name;
        }

        @Override
        public void SetDelay(int value) {
                this.eet = value;
        }

        @Override
        public void SetDelayInRange(int eet, int let) {
                this.eet = eet;
                this.let = let;
        }

        @Override
        public String Print() {
                return this.Name + "\n" + this.PH.PrintAllPlaces();
        }
}
```

Code sequence 12: Child transition t_4

```
import Interfaces.PlaceHandlerTemplate;
import Interfaces.TransitionTemplate;

public class Child_Transition_t_4 implements TransitionTemplate {
        Integer timeUnitControl = 500;
        Integer eet;
```

```
	Integer let;
	String Name;
	PlaceHandlerTemplate PH;
	Child c;

	public Child_Transition_t_4(String name, PlaceHandlerTemplate PH, Integer delay) {
		this.Init(name, PH);
		this.SetDelay(delay);
	}

	public Child_Transition_t_4(String name, PlaceHandlerTemplate PH, Integer eet, Integer let)
{
		this.Init(name, PH);
		this.SetDelayInRange(eet, let);
	}

	@Override
	public void TransitionDelay() {
		try {
			if (this.let == null) {
				Thread.sleep(this.eet * timeUnitControl);
			} else {
				Thread.sleep(Math.round(Math.random() * (this.let - this.eet) + this.eet)
* timeUnitControl);
			}
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	@SuppressWarnings("deprecation")
	@Override
	public Boolean TransitionGuardsMappings() {
		TransitionDelay();
		String toPrint = "--------------Child--------------------\n";

		if (!PH.GetPlaceByName("p_23").IsNull() && (Float)
(PH.GetPlaceByName("p_23").Get()) >= 2.0f) {

			toPrint = toPrint.concat(Print() + "\n");
			toPrint = toPrint.concat("ChildrPH\n" + PH.PrintAllPlaces() + "\n");

			System.out.println("The thread is stopped, the state is passive");
			c.StopThread();
			toPrint = toPrint.concat(Print() + "\n");
			toPrint = toPrint.concat("ChildPH\n" + PH.PrintAllPlaces() + "\n");
			toPrint = toPrint.concat("-------------------------------------\n");
```

```
                    System.out.println(toPrint);

                    return true;
                } else
                    return false;
        }

        @Override
        public void Init(String name, PlaceHandlerTemplate PH) {
                this.PH = PH;
                this.Name = name;
        }

        @Override
        public void SetDelay(int value) {
                this.eet = value;
        }

        @Override
        public void SetDelayInRange(int eet, int let) {
                this.eet = eet;
                this.let = let;
        }

        @Override
        public String Print() {
                return this.Name + "\n" + this.PH.PrintAllPlaces();
        }
}
```

D. Developments and Tests:

App 1: Test by sending 1 to the parent's place p_4 and run the application again and test by sending 2 to p_4. Explain what happens.

App2: Draw the Class diagram and the state machines diagrams for the Multi-Tasking Implementation of example 4 .

References:

[1] T. Letia and D. Al- Janabi, "Object Enhanced Time Petri Nets Models", IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), DOI: 10.1109/AQTR.2018.8402743, 2018.