RIPN system construction rules. v1.3.2

25th June 2025   *Pedro Sánchez Palma*

pedro.sanchez@upct.es

Universidad Politécnica de Cartagena, Spain

For a full understanding of the approach around RIPN, we encourage you to read the following paper

Code available at: https://github.com/ppalma00/RIPN

Summary:

RIPN is a Java-based tool that allows for the integration of a specific type of interpreted Petri net with teleo-reactive (TR) programs. The Petri net is interconnected with the environment such that the occurrence of events in said environment serves as input for triggering transitions, which we call of type "input" Transitions can also have conditions defined on a set of variables (integers or real-world variables) and/or facts, all managed in a BeliefStore shared with the TR program. A RIPN system can be defined by a TR program, a Petri net, or a combination of both, such that changes in the BeliefStore are shared between both subsystems. Thus, the Petri net influences the TR program and vice versa.

The system allows for the execution of discrete or durative actions, both in the Petri net and in the TR program. In the TR program, these actions are executed in the rules. In the Petri net, they are executed only in the places when they acquire the marking. Changes to the BeliefStore (changing variables or adding/deleting facts) can be made in both places and transitions. Actions and facts can have parameters. Timers can be added and manipulated (paused, continued, stopped). The end of a timer is automatically added to the BeliefStore.

The semantics associated with durative and discrete actions are the same for both the TR program and the Petri net. The only consideration is that for the Petri net, the reference will be the marking of a previously unmarked place, while for the TR program, the reference will be the activation of a rule not active in the immediately preceding state.

The system allows you to leave traces of everything that happens in different files with time stamps to facilitate debugging.

Key features

- RIPN allows the concurrent execution of binary interpreted Petri nets and Teleo-Reactive programs.
- Petri nets and TR programs share a BeliefStore where facts (with or without parameters) and variables (INT or REAL) are stored.
- There is one thread of execution for Petri nets, another one for TR programs and a third one for the management of the pool of events received by the system. The events affect certain transitions of the Petri nets and can wait to be attended indefinitely or have a deadline.
- Both Petri nets and TR programs can execute discrete or durative actions (on the environment). The former are triggered whenever a location gains a marker (for the Petri net) or a rule becomes active and was not (for the TR program). Durative actions are executed while the place is marked, or the rule is active. Both consider the use of parameters.
- It is possible to manage timers that are started ("timer.start", indicating seconds as parameter), can be paused, continued or stopped. The end of a timer is an event that is added to the BeliefStore ("timer.end").
- It is possible to change from "main" the refresh time of the Petri net or TR program status. It is also possible to set whether to have a log on screen or in a file on disk (one for the Petri net and one for the TR program).
- Petri net transitions are divided between immediate (all those that do not appear in the <PN> specification block along with those that do appear but are not affected by external events) and the so-called "input", which are the non-immediate ones (all those that appear in the <PN> block and have a "when" clause). The system evolves as long as there are immediate ones to be triggered. When a stable state is reached then the non-immediate ones that can be triggered are analyzed (the event associated with them is available in the event pool).
- Durative or discrete actions as well as changes in the BeliefStore (changes in variables with arithmetic expressions, adding a fact, deleting a fact or all occurrences of a fact using the wildcard character "_") can be executed at Petri net places. A condition can be specified on BeliefStore variables or facts to limit the execution of such changes using the "if" statement.
- In Petri net transitions, changes on the BeliefStore can be made but no actions can be performed or initiated. Changes to the BeliefStore can be limited to the fulfillment of a condition (indicated by "if") and the occurrence of an event (indicated by "when") for the case of non-immediate transitions. The possible parameters indicated in the event in the "when" clause serve as input values for the changes on the BeliefStore indicated in that transition when triggered.
- The execution of the Petri net and the TR program is concurrent so that one affects the other by the changes it carries out on the BeliefStore or, indirectly, by the changes it carries out on the environment with the execution of the actions. It is possible from the TR program to send an event to the environment that will be captured by the event manager and subsequently processed by the Petri net.

Concerning the TR programs

The template is the following (elements of lists are separated by ";"):

FACTS: list of facts with or without parameters (INT, REAL)
PERCEPTS: list of percepts with or without parameters (INT, REAL)
VARSINT: list of INT vars
VARSREAL: list of REAL vars
INIT: initialization of vars and facts
DISCRETE: list of discrete actions with or without parameters
DURATIVE: list of durative actions with or without parameters
TIMERS: list of timers
PLACES: list of places
TRANSITIONS: list of transitions
ARCS: list of arcs with the syntax: place->transition or transition->place, or place-o>transition
INITMARKING: tuple of '0's and '1's with the initial marking, like (1,0,0,01)
<TR>
condition -> list of actions(discrete, durative and timers) [operations on the BeliefStore (variables and facts/percepts)]

As with the Petri nets, INT or REAL variables are considered. A section is included first, before the rules, where the following statements are made:

FACTS: statement of facts, indicating whether they have parameters or not, e.g., open_clamp, see_robot(INT, INT), all separated by ';'. They shall be added using remember, e.g. remember(see_robot(5,6)), and deleted using forget, e.g. forget(open_clamp), or forget(see_robot(2,3)) or forget(see_robot(_,5)), where all the occurrences matching the pattern are being removed.

- **PERCEPTS:** This section allows you to declare the "percepts" to which the TR program will react. They are treated the same as "facts" for all purposes (conditions, and forget/remember operations). The only difference is that their activation comes from the TR program's perception of its environment, as discussed later. The tool will generate a GUI for emulating the environment for these percepts.

- **VARSINT:** declaration of integer variables separated by ';'.

- **VARSREAL:** declaration of real variables separated by ';'.

- **DISCRETE:** discrete actions, with or without parameters, e.g. open(INT, INT), close().

- **DURATIVE:** durative actions, with or without parameters, same syntax as discrete, e.g. rotate(), move(REAL).

- **TIMERS:** declaration of timers, e.g. timer1. Timers can be started, paused, continued or stopped. The ending of a timer adds the fact "*name_of_the_timer*.**end**" to the BeliefStore.

- **INIT:** initial values to integer or real variables using the symbol ":=". For example: x:=1, y:=5.66. If not specified, a value of 0 is assumed. It is possible to initialize the BeliefStore adding at once many instances of a fact. For instance, for the fact "see(INT)" it is possible at the INIT section to do "see(1..100)", then adding automatically the fact instances "see(1)", ..., "see(100)" to the BeliefStore. At the same time, in a rule many all the occurrences can be removed by doing "forget(see(_))" using the "_" character.

The declaration section is followed by the block delimited by the "<TR>" mark where the rules are specified, from highest priority (top) to lowest. The rules are constructed with the format:

```
<TR>
condition -> action(s) [changes in the BeliefStore, separated by ';']
```

Conditions are logical expressions with the usual operators (&&, ||, !) using BeliefStore variables and facts. When using facts with parameters in rule conditions, there are two possible situations:
1. The variable (or variables in plural if there is more than one) is included with the "**out**" modifier. The interpreter then looks for an instance of that fact that matches the pattern given for the rest of the parameters and initializes the indicated variable with the corresponding value of the fact in the BeliefStore. For example, if the fact "see(4)" exists in the BeliefStore and a condition has the form "see(**out** x)", then x will take the value 4 and the condition will be true. Assigning x to 4 is like any other initialization made to a variable in the TR program.
2. The variable is included without the "**out**" modifier. The interpreter then takes the current value of that variable and checks if an instance of the fact with that value exists. The condition will be true in that case, or false otherwise.

It is possible to include the non existence of a specific fact in the BeliefStore like:
        !fact2(4) -> discreteAction(4)

And by using the current value of variables:
        !fact2(z) -> discreteAction(4)

Or even to check there is no instance of a fact, like:
        !fact2(_) -> discreteAction(4)

The use of "out" provides greater richness for expressing different situations. For example, if we have the following rule:
        fact2(_, **out** s) -> discreteAction1(s)
then, if the fact "fact2(4, 7)" is in the BeliefStore, "s" will be given the value 7, and the operation "discreteAction1(7)" will be executed. If the fact "fact2(4, 5)" is in the BeliefStore, then the matching does not occur, and the variable "s" is not modified. Another example:

FACTS: see(INT, INT)
PERCEPTS:
VARSINT: x; y
VARSREAL:
DISCRETE: do(INT)
DURATIVE:
TIMERS:
INIT: x:=1; y:=3

<TR>
see(**out** y, _) -> do(y)
x==1 -> do(x) [x:=x+1; remember(see(100,2))]

In this example, the first parameter is of type output, and the second uses a wildcard character, indicating that any value is valid. The trace of this execution is as follows (highlighted in red color the execution of the highest priority rule):

ℹ️ Initialized integer variable 'x' to 0
ℹ️ Initialized integer variable 'y' to 0
Initiating tr program...
[2025-06-13 13:04:02.117]
🔷 Current BeliefStore state:
   Active facts without parameters: []

Active facts with parameters: {}
Integer variables: {x=1, y=3}
Real variables: {}

[2025-06-13 13:04:02.133] 🔄 Executing rule with condition: x==1
[2025-06-13 13:04:02.135] ▶️ Executing discrete action: do with parameters: [1.0]
[2025-06-13 13:04:02.139] Observer: Executing discrete action: do with parameters: [1.0]
[2025-06-13 13:04:02.239]
🔷 Current BeliefStore state:
Active facts without parameters: []
Active facts with parameters: {see=[[100, 2]], }
Integer variables: {x=2, y=3}
Real variables: {}

[2025-06-13 13:04:02.247] 🔄 Executing rule with condition: see(out y, _)
[2025-06-13 13:04:02.247] ▶️ Executing discrete action: do with parameters: [100.0]
[2025-06-13 13:04:02.247] Observer: Executing discrete action: do with parameters: [100.0]
[2025-06-13 13:04:02.351]
🔷 Current BeliefStore state:
Active facts without parameters: []
Active facts with parameters: {see=[[100, 2]], }
Integer variables: {x=2, y=100}
Real variables: {}

However, if we have the following rule then the current value of "s" is used to verify the condition and the execution of the action.

        fact2(_, s) -> discreteAction1(s)

Each rule can include none, one or more actions, whether durative or discrete, using in its argument's values from the BeliefStore variables or arithmetic expressions with the usual operators (+, -, *, /).

Changes in the BeliefStore can be changes to variables with the usual arithmetic expressions (+,-,*,/) and/or adding or deleting facts (remember/forget), with or without parameters. It is possible to use the wildcard character "_" in the facts both in the conditions and in the "forget" operation (which would delete all occurrences matching the indicated pattern).

Declared timers can be started ("start", with seconds in brackets), paused ("pause"), continued ("continue") or stopped ("stop") as a discrete action. When a timer ends, the fact ".end" is automatically added to the BeliefStore, can be used in the conditions and can be removed manually, "forget(t1.end)" in the case of a timer named "t1".

For example:

```
FACTS: seen(INT); done
PERCEPTS:
VARSINT: x; y;
VARSREAL: z
DISCRETE: open(INT, REAL)
DURATIVE: rotate(INT); close() TIMERS: t1
INIT: x:= 0; z:=4.5

<TR>
seen(3) && x==2 && t1.end-> open(2, 4.5); rotate() [forget(seen(_)]] seen(_) && x==4 -> open(3, 6.6)
y<5 -> t1.continue(); close() [x:=3]; y<5 -> t1.continue(); close() [x:=3]. x> 1 -> rotate(3); close()
[y:=4; remember(seen(3))]
x==0 -> t1.start(4)
True -> act1() [y:=2; z:=z+1]
```

The system is continuously evaluating the highest priority rule to be executed. There must always be at least one that meets the condition (hence the lowest one is usually set to True).

Durative actions are running while the rule that initiated them is active. Discrete actions are triggered upon each activation of the rule. It requires another rule to gain control and the first rule to re-activate for the execution of the discrete action to be triggered again. Operations on timers are considered discrete. Starting a timer that is already running removes the previous instance of the timer.

Another example:
FACTS:
PERCEPTS: see(INT)
VARSINT: x; y; z
VARSREAL:
DISCRETE: do(INT); do2(INT)
DURATIVE:
TIMERS:
INIT: x:=1; y:=3; see(100)

<TR>
see(out z) && z > 150 -> do2(z)
x==1 -> do(x) [x:=x+1; remember(see(200))]

In this case, the valid value for the variable "z" is 200 and not 100 so, given "see(200)" as a fact existing in the BeliefStore, z will be 200 once the lower priority rule has been executed and "see(200)" has been added to the BeliefStore.

Another example:

FACTS: veo(INT); touch
PERCEPTS:
VARSINT: x; y; z
VARSREAL:
DISCRETE: act1(); act2()
DURATIVE: actFIN()
TIMERS: t1
INIT: x:=0

<TR>
t1.end -> actFIN()
x==2 -> act2() [remember(touch)]
True -> act1(); t1.start(5) [y:=2; z:=55; x:=2; remember(veo(8))]

In this TR program, one fact is declared with arguments and another ("touch") without arguments (note that parentheses are not included, unlike actions, where they are included by convention). The rule below activates a 5-second timer and executes the discrete action "act1()". In addition, several changes are made to the BeliefStore. The change in "x" implies that the rule with the discrete action "act2()" is executed, and the fact "touch" is remembered. After 5 seconds have elapsed, the durative action "actFIN()" is executed. The trace generated from this execution is as follows:

2025-05-07 10:51:15.936]
◆ Current BeliefStore state:
   Active facts without parameters: []
   Active facts with parameters: {}
   Integer variables: {x=0, y=0, z=0}

Real variables: {}

[2025-05-07 10:51:15.951] 🔄 Executing rule with condition: True
[2025-05-07 10:51:15.952] ⏩ Executing discrete action: act1 with parameters: []
[2025-05-07 10:51:15.952] ⏩ Executing discrete action: act1 with parameters: []
[2025-05-07 10:51:15.957] ⏩ Executing discrete action: t1.start with parameters: [5.0]
[2025-05-07 10:51:15.958] 🛠 Extracted timer command: start for timer: t1
[2025-05-07 10:51:15.958] ⏳ Timer started: t1 for 5 seconds
[2025-05-07 10:51:16.072]
🔷 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {veo=[[8]], }
  Integer variables: {x=2, y=2, z=55}
  Real variables: {}

[2025-05-07 10:51:16.076] 🔄 Executing rule with condition: x==2
[2025-05-07 10:51:16.076] ⏩ Executing discrete action: act2 with parameters: []
[2025-05-07 10:51:16.077] ⏩ Executing discrete action: act2 with parameters: []
[2025-05-07 10:51:16.177]
🔷 Current BeliefStore state:
  Active facts without parameters: [touch]
  Active facts with parameters: {veo=[[8]], }
  Integer variables: {x=2, y=2, z=55}
  Real variables: {}

[2025-05-07 10:51:16.282]
🔷 Current BeliefStore state:
  Active facts without parameters: [touch]
  Active facts with parameters: {veo=[[8]], }
  Integer variables: {x=2, y=2, z=55}
  Real variables: {}

The output shown in the console (which acts as an observer object for the TR program outputs) is as follows:

Initiating tr program...

[2025-05-07 10:51:15.955] Observer: Executing discrete action: act1 with parameters: []

[2025-05-07 10:51:16.077] Observer: Executing discrete action: act2 with parameters: []

[2025-05-07 10:51:21.075] Observer: Starting durative action: actFIN() with parameters: []

The tool considers a discrete action called "**_send**" which receives as first argument the name of the event to be sent to the environment (for further processing from the Petri net side), followed by the actual values of the event parameters. For instance, doing in a rule **_send**("open", 3, 45)" will result in adding the event "open(3,45)" to the pool of events pending of processing by the Petri net.

With this example:
FACTS:
PERCEPTS: see(INT)
VARSINT: x; y; z
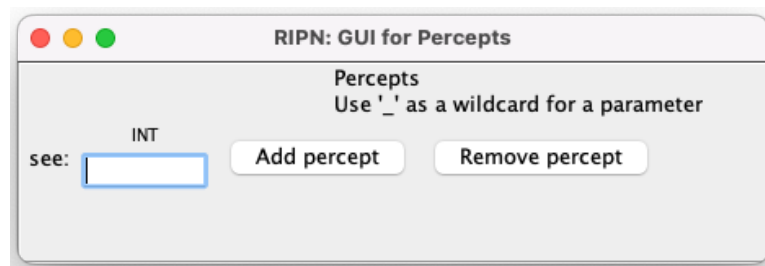VARSREAL:
DISCRETE: do(INT); do2(INT)
TIMERS: timer1
INIT: x:=1; y:=3; z:=4; see(5)

<TR>

!see(_) -> do2(z); timer1.pause()
x==1 -> do(x); timer1.start(6) [x:=x+1; forget(see(_))]

The tool automatically pops up the Java Swing interface for providing or deleting percept instances:

The trace with the execution is the following with no percept given with the interface:

ℹ️ Initialized integer variable 'x' to 0

ℹ️ Initialized integer variable 'y' to 0

ℹ️ Initialized integer variable 'z' to 0

Initiating tr program...

[2025-06-27 11:07:51.270]

🔷 Current BeliefStore state:

  Active facts without parameters: []

  Active facts with parameters: {see=[[5]], }

  Integer variables: {x=1, y=3, z=4}

  Real variables: {}

[2025-06-27 11:07:51.293] 🔄 Executing rule with condition: x==1

[2025-06-27 11:07:51.295] ⏭️ Executing discrete action: do with parameters: [1.0]

[2025-06-27 11:07:51.300] Observer: Executing discrete action: do with parameters: [1.0]

[2025-06-27 11:07:51.301] 🛠️ Extracted timer command: start for timer: timer1

[2025-06-27 11:07:51.301] ⏳ Timer started: timer1 for 6 seconds

[2025-06-27 11:07:51.301] 🔍 Calling removeFactWithWildcard with: see(_)

[2025-06-27 11:07:51.302] 🗑️ Removed facts matching wildcard pattern: see(_)

[2025-06-27 11:07:51.406]

🔷 Current BeliefStore state:

  Active facts without parameters: []

  Active facts with parameters: {}

  Integer variables: {x=2, y=3, z=4}

  Real variables: {}

[2025-06-27 11:07:51.406] 🔄 Executing rule with condition: !see(_)

[2025-06-27 11:07:51.406] ⏭️ Executing discrete action: do2 with parameters: [4.0]

[2025-06-27 11:07:51.406] Observer: Executing discrete action: do2 with parameters: [4.0]

[2025-06-27 11:07:51.407] 🛠️ Extracted timer command: pause for timer: timer1

[2025-06-27 11:07:51.407] ⏸️ Timer paused: timer1, remaining time: 5894 ms

Concerning the Petri Nets

The template is the following (elements of lists are separated by ";"):

FACTS: list of facts with or without parameters (INT, REAL)
VARSINT: list of INT vars
VARSREAL: list of REAL vars
INIT: initialization of vars and facts
DISCRETE: list of discrete actions with or without parameters
DURATIVE: list of durative actions with or without parameters
TIMERS: list of timers
PLACES: list of places
TRANSITIONS: list of transitions
ARCS: list of arcs with the syntax: place->transition or transition->place, or place-o>transition
INITMARKING: tuple of '0's and '1's with the initial marking, like (1,0,0,01)
EVENTS: list of events. The first parameter is the duration of the event in the pool.
this is a comment
<PN>
place: [*operations on the BeliefStore or actions*] if(*condition*)
transition: when(*event*) [*operations on the BeliefStore*] if(*condition*)

For places, the condition part is optional. For transitions, the condition part and the when part are optional. Transitions with the "when" part are non-immediate.

The initial declaration section is the same as for the TR program with the same considerations for facts, variables, initialization, actions, and timers. Each part of the system (Petri net or TR program) must declare what it uses in its context. If a variable/fact/action/timer is declared in both the Petri net and the TR program, it is because it is the same variable/fact/action/timer, so that changes made by one part of the system are visible from the other part.

One difference is that in TR programs, we have a PERCEPTS section, which is not present in Petri nets, and in Petri nets, we have an EVENTS section, which is not present in TR programs. Thus, the input interaction from the environment with TR programs is the perceptions it has of it, and the input interaction to the Petri net is through events, which can also be received from a TR program.

RIPN considers **binary Petri nets**, i.e., a place can hold at most one token.

After the part of the basic declarations, a Petri net part is included where places, transitions, arcs and initial marking are indicated. Places, transitions and arcs are separated by ";". Places and transitions must have disjoint identifiers.

An arc is indicated by "place->transition". If the arc is inhibiting, "-0>" is the syntax. The specification of the arcs can be split into several entries in the file if necessary (each on a separate line starting with "ARCS"). The initial marking must give a value (0 or 1) to each place considering the same order as given in the PLACES declaration section. For example:

FACTS:
VARSINT: x; y
VARSREAL:
INIT:
DISCRETE: open(INT, REAL)

DURATIVE:
TIMERS: t1
PLACES: p0; p1
TRANSITIONS: t0; t1
ARCS: p0->t0; t0->p1; p1->t1; t1->p0

INITMARKING: (1,0)

EVENTS:

<PN>
p0: [open(3,4); t1.start(8); x:=1]
t0: [] if(y==2)
p1: [x:=2]

The EVENTS section allows you to specify which events will be attended to among those occurring in the environment, thereby enabling the triggering of certain transitions. The framework will include this event in the graphical interface so that the user can emulate the arrival of an event to the system. The first parameter in an event declaration is the duration of how long the event will remain in the received event queue. A 0 means unlimited. The second and subsequent arguments, if indicated, correspond to the type of the expected parameters, INT or REAL. For example:

EVENTS: see(0, INT)
represents an event that remains in the event queue for an infinite amount of time and takes an integer as an argument.

EVENTS: near(10)
represents an event that will remain in the event queue for up to 10 seconds and takes no arguments.
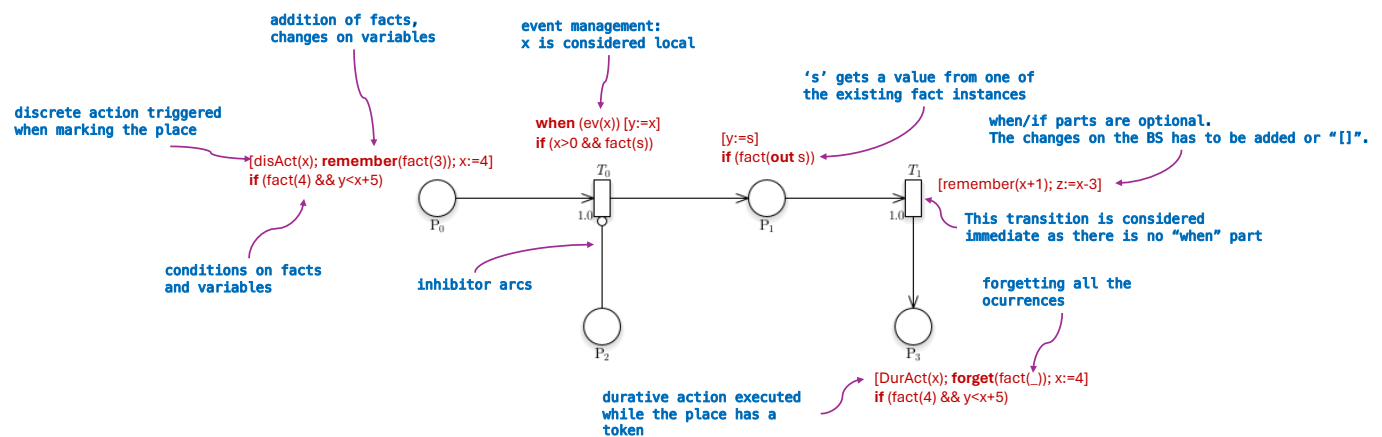
Another interesting aspect of events is their interpretation within transitions. When we have:
t0: **when**(see(x)) [y:=x; action(x)] **if** (x<10)

This means that when the "see" event occurs, the argument "x" is used to check the associated condition, allowing us to limit the transition's firing. We can also use the value returned by the event to modify the BeliefStore, initiate actions, etc.

Some further examples:

when(ev(x)) [y:=x] **if**(x>5)
'x' is temporally assigned the value that comes with the event being consumed. This value is

used in the "[]" operations and is considered in the "if" condition. If we want the value of "x" to change as a result of the event, we have to make it explicit:

when(ev(x)) [x:= x; y:=x] **if**(x>5)

Another way is to use a temporary variable for this:
when(ev(temp)) [x:=temp; y:=temp] **if**(temp==15)

[] **if**(see(s) && x<5)
's' and "x" currently have values that are used to check the condition. In the case of the fact, if "s" equals 3, for example, that part of the condition will be true if the fact "see(3)" currently exists in the database.

[] **if**(see(s) && x<5)
The variable "s" has a specific value at that time, so it will be checked if there is an instance of the "see" event with that value, and x<5.

[z:=s] **if**(see(**out** s) && x<5)
For "x", the current value in the Beliefstore is used. For "s", the first existing fact that makes the first part of the condition true is searched for, assigning "s" the value corresponding to the existing fact. The variable "z" will be assigned the value of "s". This is the same functionality we already have with the TR programs.

when(ev(x)) [y:=x]
In this case, the value of "x" is simply obtained from the event parameter and used in the assignment of the value to the variable "y".

when(ev(x)) [remember(see(x))]
In this case, following the same approach, the value of "x" from the event is used to collect the fact "see(x)".

t1: [x:=z*10] if(see(out z) && hold(z-1) && x>10)
You can use arithmetic expressions on the parameters of a fact (as with "hold" for the example), but not involving "out" prefix.

After the declarations, a block delimited by the "<PN>" mark specifies the behaviour. The changes that are carried out in the BeliefStore are indicated for the appropriate places with the same syntax that we discussed for the TR programs but also adding the actions (durative or discrete) and the handling of the timers. The conditions follow the same considerations as for TR programs:

```
place_name: [operations on the BeliefStore or durative/discrete actions/timers, separated by ";"] if
(condition on the BeliefStore)
```

Not all places need to be listed in this section, only those where changes are made to the BeliefStore and/or actions are initiated. Durative actions are running while the place is marked. Discrete actions only when the place loses the marking and regains it (same analogy as with the TR programme rules).

The transitions we call of type "input" are also specified:

```
transition_name: when (event_name) [operations on BeliefStore, separated by ";"] if(condition on
BeliefStore)
```

The operations part must be included even without any operation in it as "[]". The "when" and "if"

parts are both optional.

Transitions that are not listed in the section or don't have the "when" part are considered of type **immediate** and have priority over incoming calls. If the system has not triggered all the immediate ones enabled to do so (their entry places are marked, and the exit places have no tokens) it will continue to evolve the net. Once all the immediate transitions have been triggered, the system is ready to evaluate the triggering of the input transitions, which are those listed in the <PN> block. The system will fire the transitions that have a condition that is met given the state of the BeliefStore. In these transitions only changes can be performed on the BeliefStore, no actions can be performed, and no timers can be managed.

Let's look at an example.

```
FACTS: see(INT)
VARSINT: x; y; p1; p2
VARSREAL:
INIT: x:=3
DISCRETE:
DURATIVE: act1(INT)
TIMERS: t1

PLACES: p0; p1; p2
TRANSITIONS: t0; t1; t2; t3
ARCS: p0->t0; t0->p1; p1->t1; t1->p0; p1->t2; t2->p2; p2->t3; t3->p0
INITMARKING: (1,0,0)
EVENTS: ev1(0, INT); ev2(2, INT, INT)

<PN>
p0: [t1.start(5); act1(88)]
t0: [x:=p1+y-p2; remember(see(x+p1))] if (t1.end)
t2: when (ev2(p1, p2)) [y:=p1+p2]
p1: [x:=x+1]
```

When place 'p0' receives the tag, it starts a 5-second timer and initiates the "act1(88)" lasting action. Once the timer has elapsed, the "t1.end" condition becomes true, so transition "t1" can be triggered, having as changes in the BeliefStore the change of variable "x" and remembering the fact "see(x+p1)". Transition "t2" has an associated event, but since transition "t1" is immediate (it is not listed in the <PN> section), it has priority over "t2", so it is never executed. The execution trace can be seen below.

[2025-05-07 09:59:02.815] ⏳ Timer started: t1 for 5 seconds
[2025-05-07 09:59:02.816] 🕐 Timer t1 started for 5 seconds
[2025-05-07 09:59:02.820] Observer: Starting durative action: act1 with parameters: [88.0]
[2025-05-07 09:59:02.822] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
[2025-05-07 09:59:02.822]
🔷 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {}
  Integer variables: {p1=0, p2=0, x=3, y=0}
  Real variables: {}

[2025-05-07 09:59:03.139] 🚫 Skipped firing transition t0 (Condition not met: t1.end)
[2025-05-07 09:59:03.139] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
[2025-05-07 09:59:03.139]

◆ Current BeliefStore state:
   Active facts without parameters: []
   Active facts with parameters: {}
   Integer variables: {p1=0, p2=0, x=3, y=0}
   Real variables: {}

[2025-05-07 09:59:05.144] 🚫 Skipped firing transition t0 (Condition not met: t1.end)
[2025-05-07 09:59:05.145] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
[2025-05-07 09:59:05.145]
◆ Current BeliefStore state:
   Active facts without parameters: []
   Active facts with parameters: {}
   Integer variables: {p1=0, p2=0, x=3, y=0}
   Real variables: {}

[2025-05-07 09:59:07.149] 🚫 Skipped firing transition t0 (Condition not met: t1.end)
[2025-05-07 09:59:07.149] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
[2025-05-07 09:59:07.149]
◆ Current BeliefStore state:
   Active facts without parameters: []
   Active facts with parameters: {}
   Integer variables: {p1=0, p2=0, x=3, y=0}
   Real variables: {}

[2025-05-07 09:59:09.155] 🚫 Skipped firing transition t0 (Condition not met: t1.end)
[2025-05-07 09:59:09.155] ✅ Timer expired: t1_end activated
[2025-05-07 09:59:09.155] 🔴 Timer fully removed: t1
[2025-05-07 09:59:09.155] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
[2025-05-07 09:59:09.155]
◆ Current BeliefStore state:
   Active facts without parameters: [t1_end]
   Active facts with parameters: {}
   Integer variables: {p1=0, p2=0, x=3, y=0}
   Real variables: {}

Another example:

FACTS: see(INT)
VARSINT: x; y; p1; p2
VARSREAL:
INIT: x:=3; p1:=5
DISCRETE: act2(INT)
DURATIVE: act1(INT)
TIMERS: t1

PLACES: p0; p1; p2
TRANSITIONS: t0; t1; t2; t3
ARCS: p0->t0; t0->p1; p1->t1; t1->p0; p1->t2; t2->p2; p2->t3; t3->p0
INITMARKING: (1,0,0)
EVENTS: ev1(0, INT); ev2(2, INT, INT)

<PN>
p0: [y:=6; act2(p1)]
t0: when(ev1(p1)) [x:=p1; remember(see(y+p1))] if (p1>100)
t2: when (ev2(p1, p2)) [y:=33]
p1: [x:=x+1]

With the initial marking of the net, the discrete action "act2(5)" is executed, since "p1" is equal to 5. The system waits for the event "ev1" to occur in order to trigger the transition "t0", which will have an INT value as an argument that is stored in the variable "p1" and will be used to evaluate the condition and, if it is met, carry out the changes in the BeliefStore (in this case, assign the value of "p1" to "x" and store the fact "see" with the value of the argument after performing the sum). The trace of this execution is the following after injecting the event "ev1(200)":

[2025-05-07 10:34:20.978] Observer: Executing discrete action: act2 with parameters: [5.0]
[2025-05-07 10:34:20.984] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
[2025-05-07 10:34:20.985]
◆ Current BeliefStore state:
　Active facts without parameters: []
　Active facts with parameters: {}
　Integer variables: {p1=5, p2=0, x=3, y=6}
　Real variables: {}

⏸ No non-immediate transitions enabled at this time. Waiting...
⏸ No non-immediate transitions enabled at this time. Waiting...
[2025-05-07 10:34:37.236] Current state of the Petri Net:
p0: ○
p1: ●
p2: ○
[2025-05-07 10:34:37.236]
◆ Current BeliefStore state:
　Active facts without parameters: []
　Active facts with parameters: {see=[[206]], }
　Integer variables: {p1=200, p2=0, x=200, y=6}
　Real variables: {}

[2025-05-07 10:34:39.241] Observer: Executing discrete action: act2 with parameters: [200.0]
[2025-05-07 10:34:39.241] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
[2025-05-07 10:34:39.241]
◆ Current BeliefStore state:
　Active facts without parameters: []
　Active facts with parameters: {see=[[206]], }
　Integer variables: {p1=200, p2=0, x=200, y=6}
　Real variables: {}

⏸ No non-immediate transitions enabled at this time. Waiting...
⏸ No non-immediate transitions enabled at this time. Waiting...

Another example:

FACTS: see(INT, INT)
VARSINT: x; y; z
VARSREAL:
INIT: x:=1; see(3,4)
DISCRETE:
DURATIVE:
TIMERS:
PLACES: p0; p1
TRANSITIONS: t0; t1
ARCS: p0->t0; t0->p1; p1->t1; t1->p0

INITMARKING:(1,0)
EVENTS: ev(0,INT)

<PN>
t0: when(ev(x)) [x:=x; y:=x; remember(see(4,9))] if(see(3,4))
t1: [x:=z*10] if(see(out z, 9) && x>10)

The trace considering the input of the event "ev(15)" is the following:

ℹ️ Initialized integer variable 'x' to 0
ℹ️ Initialized integer variable 'y' to 0
ℹ️ Initialized integer variable 'z' to 0
[2025-06-20 10:33:42.729] Current state of the Petri Net:
p0: ●
p1: ○
[2025-06-20 10:33:42.729]
🔶 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {see=[[3, 4]], }
  Integer variables: {x=1, y=0, z=0}
  Real variables: {}

🚫 Transition 't1' blocked by condition: see(out z, 9) && x>10
⏸️ No non-immediate transitions enabled at this time. Waiting...
[2025-06-20 10:33:47.029] 🔥 Transition fired: t0
[2025-06-20 10:33:47.029] Current state of the Petri Net:
p0: ○
p1: ●
[2025-06-20 10:33:47.029]
🔶 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {see=[[3, 4], [4, 9]], }
  Integer variables: {x=15, y=15, z=0}
  Real variables: {}

[2025-06-20 10:33:49.038] 🔥 Transition fired: t1
[2025-06-20 10:33:49.039] Current state of the Petri Net:
p0: ●
p1: ○
[2025-06-20 10:33:49.039]
🔶 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {see=[[3, 4], [4, 9]], }
  Integer variables: {x=40, y=15, z=4}
  Real variables: {}

⏸️ No non-immediate transitions enabled at this time. Waiting...

It can be seen that the "out" prefix for the "z" variable at "see" fact is interpreted as extracting the current value of a existing fact in the BeliefStore to instantiate the value of, "z", new value which can be used to do further operations on variables (like x:=z*10). The same example but with this code for "t1" transaction:

        t1: [x:=z*10] if(see(z, 9) && x>10)

it means that "t1" needs the fact "see(0,9)" in the BeliefStore (as "z" is equals to 0). In this case,

the condition will evaluate to false.

Concerning Event Inputs for the Petri net

From the Java Swing interface, we can inject events into the system as simulations. Events are declared in the corresponding section:

EVENTS: ev1(0, INT); ev2(2, INT, INT)

As said before, the first parameter indicates how many seconds the event will be active in the queue of events waiting to be served. A '0' indicates that there is no time limit. Once the time has passed, if the event has not been consumed since any Petri net transition, it is removed from the pool. The history of occurrences is kept in the trace. If an event is received with an incorrect number of parameters, it is rejected and recorded in the trace. A graphical interface is provided to manage event injection. In a real deployment, it would be replaced by the corresponding Java module to receive events from the environment. The tool automatically generates as many inputs in the interface as events declared, and as many text boxes as parameters declared. Type checking is also performed to avoid sending an unwanted value. It is possible from this interface to delete all events from the pool.
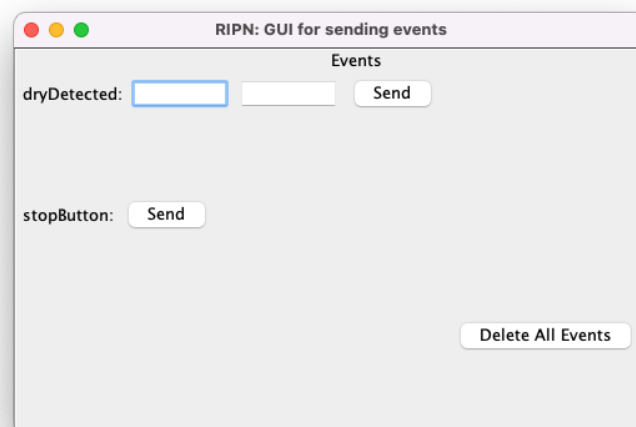
A possible trace could be the following where it can be observed that the ev2 expires after 2 seconds and that it was received at the end with an incorrect number of parameters.

```
[LOG STARTED] 2025-05-07 09:55:44.081
[2025-05-07 09:55:52.174] Event added: ev1
Active Events:
    ev1 [100.0] expires in 9223370290250223633 ms
Unattended Events:

[2025-05-07 09:55:58.747] Event added: ev2
Active Events:
    ev2 [200.0, 100.0] expires in 1999 ms
    ev1 [100.0] expires in 9223370290250217060 ms
Unattended Events:

[2025-05-07 09:56:02.348] Expired events cleaned
Active Events:
    ev1 [100.0] expires in 9223370290250213459 ms
Unattended Events:

[2025-05-07 09:56:02.351] Event wrong arity: ev2
Active Events:
    ev1 [100.0] expires in 9223370290250213456 ms
Unattended Events:
    ev2[200.0] (wrong arity)
```

Regarding the outputs

The system provides for three output files whose names can be specified in the class included in MainUnified.java:

•   One corresponding to a dump of the state of the BeliefStore by the Petri net and the TR program, each at the completion of each execution cycle.

•   One corresponding to the execution trace of the TR program, indicating the rules that are being activated and possible errors detected both in the loading of the TR program from the input file and in the execution.

•   One corresponding to the evolution of the Petri net marking and possible incidents that may occur during implementation.

•   One corresponding to the evolution of the event pool.

It is possible to have all outputs displayed by indicating it in the main program.

It is also possible to run only the Petri net or only the TR program for debugging purposes with outputs in the console, for this purpose, we have included two files with the main method for each part.

By default, it is in main where the outputs resulting from the execution as a RIPN system are shown: discrete actions triggered and durative actions started or stopped. All other things that happen in the system are plotted in the files mentioned above because they do not correspond to the effect of the system execution. For a real system deployment, those messages that are received in main as the observer object that it is, must be passed on to the corresponding physical system devices and actuators, making the appropriate modifications to the code. Having decoupled it following the observer pattern, the change is as simple as passing a different observer to both the network and the TR program instead of being the object included in the main method.

Architecture of the RIPN framework in Java

The program has been developed in Java and can be extended as desired. The packages are shown below, with details of the role played by each of the included Java classes. The PN subpackage for Petri nets includes the MainPN.java class, which includes the main method for running a Petri net separately. The same applies to the TR package, which allows a TR program to be run in isolation. This can facilitate testing before integrating the entire system.
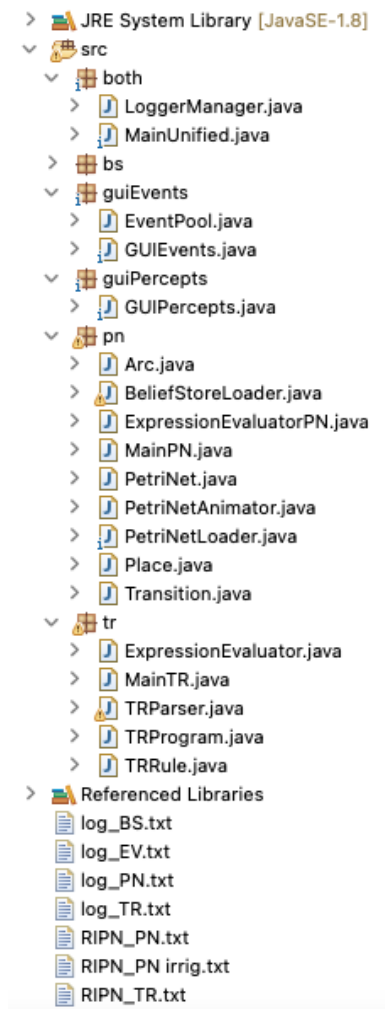
The template for the PN specification is as shown in Table 1. Same for the TR specification in Table 2.

Section	Observations
FACTS	The declaration of the facts. They can include parameters, INT or REAL. When removed from the places or transition firing, the character "_" will denote whatever value, for instance, by doing see(4,_) will delete all the occurrences of the "see" fact including as first parameter the number 4, not considering the second parameter.
VARSINT	The declaration of INT variables.
VARSREAL	The declaration of REAL variables.
INIT	The initial values for facts, INT and REAL variables.
DISCRETE	The declaration of the discrete actions. They can include parameters, INT or REAL.
DURATIVE	The declaration of the durative actions. They can include parameters, INT or REAL.
TIMERS	The declaration of timers. They can be started (indicating the duration in seconds), paused, continued, or stopped. When a timer ends, a fact is automatically added to the BeliefStore to indicate this.
PLACES	Declaration of the places.

TRANSITIONS	Declaration of the transitions.
ARCS	Normal and inhibitor. They always take a single token.
INITMARKING	Init marking for the places.
EVENTS	Events which may attend the PN. All of them include at least a integer number to indicate how long (seconds) the event will be in the queue (a value equal to 0 is interpreted as always). These events will be consumed by the non-immediate transitions.
<PN>	The last section serves to specify the behaviour of the PN regarding places and transitions: - Effect of the marking of the places: operations (on variables or remember/forget of facts), the execution of discrete actions (when a place gains a token and was previously empty) and durative actions (while the place has a token). These operations can have associated a condition defined on the BeliefStore state. - Effect of the firing of the transitions: it includes a "when" part (optional) to specify the event needed to fire the transition (the arguments are indicated with variables, which brings data to the system from the environment), a set of operations (optional) on the BeliefStore (variables and facts), and a condition (optional). Those transitions without the "when" part or not included in the <PN> section, are considered immediate.

Table 1. Parts of a PN specification.

Section	Observations
FACTS	
VARSINT	
VARSREAL	
INIT	Same as for the PN.
DISCRETE	
DURATIVE	
TIMERS	
PERCEPTS	Declaration of percetps.
<TR>	The last section serves to specify the behaviour of the TR including the rules. For each rule, the user will provide the condition (on the variables and facts of the BeliefStore) followed by the actions to be executed and the operations on the BeliefStore (same syntax as with the PN). Actions and operations are optional for a rule but at least one of them must be given.

Table 2. Parts of a TR specification.

Java Class Responsibilities in the TR Program Execution Framework

The TR (Teleo-Reactive) program execution system is composed of several coordinated Java classes, each responsible for a specific part of the program parsing, rule evaluation, action execution, and belief state management. The following describes the main roles of each class in the system:

1. MainTR.java
This class serves as the entry point for executing a TR program. It is responsible for initializing the BeliefStore, parsing the input TR file, configuring the cycle delay, and starting the main execution thread of the TRProgram. It ensures the system remains active, allowing the TR program to continuously evaluate and respond to changes in the belief state.

2. TRParser.java
The TRParser class is responsible for parsing the TR program file. It reads the declaration blocks (facts, variables, actions, timers, and initial values) and constructs the corresponding internal representations. Additionally, it extracts the TR rules and builds a TRProgram instance that contains all parsed rules and metadata.

3. TRProgram.java
This class encapsulates the TR program logic. It holds the list of rules (TRRule instances), the belief store, timer management, and the execution cycle. The run() method in this class defines the reactive loop: it continuously evaluates the rules based on the current belief state and executes the first applicable rule, updating the state accordingly.

4. TRRule.java
Each TR rule is represented as an instance of the TRRule class. It stores:
- A logical condition (evaluated against the belief state),
- A list of discrete actions to be executed when the rule fires,
- An optional list of updates to be applied to the BeliefStore.
This class provides methods for evaluating whether the rule is applicable and for executing its associated actions.

5. BeliefStore.java
The BeliefStore represents the dynamic state of the system. It maintains the currently active facts (both parameterized and unparameterized), as well as declared integer and real variables. It provides methods for:
- Declaring and checking facts or variables,
- Retrieving and setting variable values,
- Remembering or forgetting facts,
- Dumping the current belief state for logging purposes.

6. ExpressionEvaluator.java
This utility class handles the evaluation of logical expressions and assignments used in rules. It relies on a dynamic expression language (e.g., MVEL) to parse and evaluate rule conditions and variable updates at runtime.

"PN" Package class roles

1. MainPN.java
This is the entry point for executing the Petri Net system in standalone mode. It:

- Loads a Petri Net specification from a file.
- Initializes the associated BeliefStore and loads variables, facts, and actions.
- Starts a PetriNetAnimator thread to execute the simulation cycle.

2. PetriNet.java

This class represents the entire Petri Net model. It contains:

- All Place and Transition objects, along with arcs.
- Logic for checking if transitions can fire, executing transitions, and updating the state (especially marking and variable changes).
- Handling of durative and discrete actions, including interactions with Observer and BeliefStore.

3. PetriNetLoader.java

Parses the input Petri Net file to:

- Create places, transitions, arcs, and initial marking.
- Associate logic and actions (e.g., conditionals, when event triggers) with places and transitions.
- Register declared events and their specifications in EventPool.

4. PetriNetAnimator.java

Controls the execution loop:

- Repeatedly checks for enabled transitions and fires them according to priority (immediate first).
- Evaluates transition conditions (if) and event triggers (when) using data from BeliefStore and EventPool.
- Coordinates with the Observer to handle actions and logs the system state periodically.

5. BeliefStoreLoader.java

Handles loading of:

- Declared facts and variables (integers and reals).
- Initial variable assignments.
- Action declarations (discrete and durative).
- Event specifications for integration with the EventPool.

6. ExpressionEvaluatorPN.java

Evaluates logical conditions and arithmetic expressions in the context of a Petri Net:

- Used to compute expressions inside transition if and action [...] blocks.
- Supports parameter substitutions from events or variable lookups in BeliefStore.

7. Place.java

Represents a place in the Petri Net:

- Holds a token state (true or false).

- Used to determine the enabled state of transitions and to track marking.

8. Transition.java

Represents a transition:

- Stores its name, associated event triggers (when) and parameter variable bindings.

- Maintains a list of variables that are populated when an event with parameters is consumed.

- Used for checking eligibility for firing in PetriNetAnimator.

9. Arc.java

Represents a directed arc:

- Links places to transitions and vice versa.

- Supports topology validation and firing logic.

"BS" package class roles

BeliefStore.java

The BeliefStore class functions as the central knowledge base for both the Petri net and the Teleo-Reactive (TR) program. It maintains the current state of symbolic and numeric information in the system. Key responsibilities include:

- **Variable Management**: Supports declaration and dynamic assignment of integer (int) and real (double) variables. Each variable is stored with its current value and can be queried or updated.

- **Fact Management**: Manages both parameterless facts (simple flags) and parameterized facts (e.g., see(2)), allowing logic-driven conditions to be evaluated against the system state.

- **Action Registration**: Differentiates between discrete and durative actions, storing which actions are declared and available in the system.

- **Timer Support**: Enables declaration, activation, and expiration tracking of timers, which can be queried in logical expressions (e.g., t1.end).

- **State Dumping**: Provides a dumpState() method for outputting the current internal state, used for debugging or trace logging.

- **Thread-Safety**: Synchronization is used where necessary to support concurrent access from multiple threads (e.g., the Petri net thread and the TR thread).

Observer.java

The Observer interface defines a contract for external components to receive notifications about action executions during simulation. It allows the system to be extended or monitored in real time. The methods include:

- onDiscreteActionExecuted(String actionName, double[] parameters): Called when a discrete action is triggered.

- onDurativeActionStarted(String actionName, double[] parameters): Called when a durative action starts execution.

- onDurativeActionStopped(String actionName): Called when a durative action is stopped due to a place losing its marking.

This interface decouples the core logic from the external response to actions, allowing flexible extensions such as GUI updates, logging, or integration with physical systems.

"GuiEvents" package class roles

Class: EventPool

The EventPool class provides a centralized, thread-safe mechanism for managing event instances within the system. It implements the singleton design pattern to ensure only one global instance is accessible throughout the application. Events are first registered with metadata (duration and parameter types) and can later be added from external sources like the graphical interface.

Each active event is stored with its parameters and an expiration time, depending on its declared time-to-live (TTL). The pool also supports event consumption (with automatic removal), removal of expired events, and tracking of invalid or unrecognized events (e.g., due to undeclared names, wrong arity, or type mismatches). All significant interactions, including additions, deletions, and expirations, are logged to a file (log_EV.txt) with timestamps.

The pool also offers maintenance features such as clearing all events or unattended ones and printing the list of declared events for debugging or audit purposes. Two nested classes, EventSpec and EventInstance, encapsulate the specification and runtime instance details respectively.

Class: GUIEvents

The GUIEvents class implements a simple Java Swing-based graphical user interface (GUI) for injecting external events into the system at runtime. It creates a windowed form that allows the user to specify an event name and up to three optional parameters (of numeric type), and to send the event to the system.

Each row in the grid layout contains fields for event name and parameters, along with a "Send" button. When pressed, the button parses the input values and passes the resulting data to the EventPool. There is also a dedicated "Delete All Events" button for clearing the entire pool.

This interface is useful for simulating interactions or conditions during execution and facilitates manual testing and validation of event-driven behavior in the system.

"Both" package class roles

LoggerManager.java

Purpose:
LoggerManager is a centralized utility class responsible for managing the system's logging behavior. It supports flexible output to either a file or the console and ensures timestamps are consistently included when required.

Key Features:

- **Dual Output Modes:**
The logger can write to either a file or to the screen, depending on the file_or_screen flag provided

during instantiation.

- **Timestamped Logging:**
Messages can be prepended with precise timestamps (to the millisecond) using Java's DateTimeFormatter, which is useful for tracking system behavior over time.

- **Thread Safety:**
The log method is synchronized to ensure that concurrent threads (e.g., from the Petri Net and TR program) do not interfere with each other's log entries.

- **Immediate Flushing:**
The logger flushes after every write when outputting to a file, ensuring that logs are always up-to-date even in case of system crashes.

- **Error-Resilient Initialization:**
If a log file fails to open, the constructor reports the error to the standard error stream without crashing the system.

Use Case: Each subsystem (Petri Net, TR Program, BeliefStore) uses its own instance of LoggerManager, ensuring separation of concerns and traceability across different log files such as log_PN.txt, log_TR.txt, and log_BS.txt.

MainUnified.java

Purpose:
MainUnified acts as the orchestrator of the complete RIPN system. It is responsible for launching and coordinating both the **Petri Net system** and the **Teleo-Reactive (TR) program**, and it serves as a unified **observer** for action monitoring.

Key Responsibilities:

- **Startup Configuration:**
Using ENABLE_PN and ENABLE_TR flags, users can specify which parts of the system should be launched (Petri Net, TR program, or both).

- **Shared BeliefStore Setup:**
A single BeliefStore is shared across both subsystems to facilitate synchronized state management.

- **Logger Injection:**
Three distinct LoggerManager instances (loggerPN, loggerTR, loggerBS) are created and passed to the corresponding subsystems for file-based logging of trace data.

- **Component Launch:**

  o The Petri Net is loaded from a file, its places, transitions, conditions, and actions are configured, and a GUI is launched for manual event injection.

  o The TR Program is parsed from a file and begins execution in a dedicated thread, looping indefinitely with a delay between cycles.

- **Action Observation:**
Implements the Observer interface, allowing it to receive and print notifications when discrete or durative actions are executed or stopped. These notifications are shown on the console using timestamps.

- **Error Management:**
Any exceptions during the startup of either system are logged appropriately, and a clear termination message is logged if either system encounters a fatal issue.

EXAMPLE OF RIPN SPECIFICATION

This an example demonstrating how to integrate both specifications into the execution of a single system sharing a BeliefStore.

PETRI NET GRAPHICAL REPRESENTATION
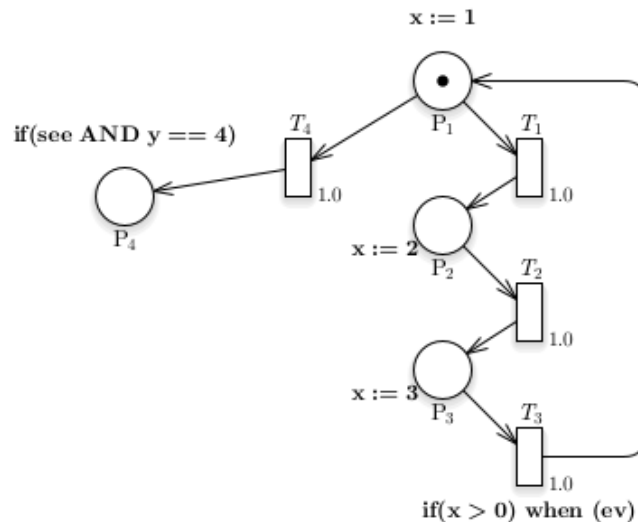
RIPN_PN.TXT file

```
FACTS: see
VARSINT: x; y
VARSREAL:
INIT: y:=0
DISCRETE: act3()
DURATIVE:
TIMERS:
PLACES: p1; p2; p3; p4
TRANSITIONS: t1; t2; t3; t4
ARCS: p1->t1; t1->p2; p2->t2; t2->p3; p3->t3; t3->p1; p3->t4; t4->p4
EVENTS: ev(0)
INITMARKING: (1,0,0,0)

<PN>
p1: [x:=1]
p2: [x:=2]
p3: [x:=3]
t4: [] if(see && y==4)
t3: when(ev) [y:=4] if (x>0)
```

RIPN_TR.TXT FILE

```
FACTS: see
VARSINT: x
VARSREAL:
DISCRETE: act1()
DURATIVE: act2(INT)
TIMERS:
INIT: x:=0

<TR>
x==3 -> act2(x)
True -> act1() [remember(see)]
```

LOG_EV.TXT FILE

[LOG STARTED] 2025-05-13 12:15:56.019
[2025-05-13 12:16:06.084] Event added: ev
Active Events:
   ev [] expires in 9223370289723409723 ms
Unattended Events:

[2025-05-13 12:16:06.236] Event consumed: ev with parameters []
Active Events:
Unattended Events:

PETRI NET LOG IN LOG_PN.TXT FILE

[2025-05-13 12:15:56.042] Current state of the Petri Net:
p1: ●
p2: ○
p3: ○
p4: ○
🚫 Transition 't4' blocked by condition: see && y==4
[2025-05-13 12:15:56.212] Current state of the Petri Net:
p1: ○
p2: ●
p3: ○
p4: ○
🚫 Transition 't4' blocked by condition: see && y==4
[2025-05-13 12:15:58.219] Current state of the Petri Net:
p1: ○
p2: ○
p3: ●
p4: ○
🚫 Transition 't4' blocked by condition: see && y==4
⏸ No non-immediate transitions enabled at this time. Waiting...
[2025-05-13 12:16:06.237] Current state of the Petri Net:
p1: ●
p2: ○
p3: ○
p4: ○
[2025-05-13 12:16:08.238] Current state of the Petri Net:
p1: ○
p2: ●
p3: ○
p4: ○
[2025-05-13 12:16:10.240] Current state of the Petri Net:
p1: ○
p2: ○
p3: ●
p4: ○
[2025-05-13 12:16:12.242] Current state of the Petri Net:
p1: ○
p2: ○
p3: ○
p4: ●
⏸ No fireable transitions at this moment. Waiting for changes...

TR PROGRAM LOG IN LOG_TR.TXT FILE

[2025-05-13 12:15:56.096] Initiating TR program...

[2025-05-13 12:15:56.114] 🔄 Executing rule with condition: True
[2025-05-13 12:15:56.114] ⏭️ Executing discrete action: act1 with parameters: []
[2025-05-13 12:15:59.131] 🔄 Executing rule with condition: x==3
[2025-05-13 12:15:59.132] ⏳ Starting durative action: act2(x)
[2025-05-13 12:16:07.161] ✅ Stopping durative action: act2(x)
[2025-05-13 12:16:07.161] 🔄 Executing rule with condition: True
[2025-05-13 12:16:07.162] ⏭️ Executing discrete action: act1 with parameters: []
[2025-05-13 12:16:11.173] 🔄 Executing rule with condition: x==3
[2025-05-13 12:16:11.173] ⏳ Starting durative action: act2(x)

BELIEFSTORE LOG IN LOG_BS.TXT FILE

[2025-05-13 12:15:56.042]
🔷 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {}
  Integer variables: {x=0, y=0}
  Real variables: {}

[2025-05-13 12:15:56.212]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {x=2, y=0}
  Real variables: {}

[2025-05-13 12:15:58.219]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {x=3, y=0}
  Real variables: {}

[2025-05-13 12:16:06.237]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {x=1, y=4}
  Real variables: {}

[2025-05-13 12:16:08.238]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {x=2, y=4}
  Real variables: {}

[2025-05-13 12:16:10.241]
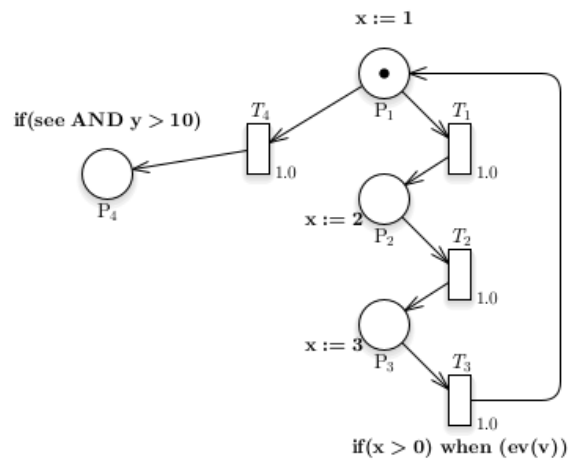🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {x=3, y=4}
  Real variables: {}

EXAMPLE OF RIPN SPECIFICATION v.2

The previous example is modified to demonstrate how to send an event from the TR program to the environment.

PETRI NET GRAPHICAL REPRESENTATION

RIPN_PN.TXT file

```
FACTS: see
VARSINT: x; y; v
VARSREAL:
INIT: y:=0
DISCRETE: act3()
DURATIVE:
TIMERS:
PLACES: p1; p2; p3; p4
TRANSITIONS: t1; t2; t3; t4
ARCS: p1->t1; t1->p2; p2->t2; t2->p3; p3->t3; t3->p1; p3->t4; t4->p4
EVENTS: ev(0, INT)
INITMARKING: (1,0,0,0)

<PN>
p1: [x:=1]
p2: [x:=2]
p3: [x:=3]
t4: [] if(see && y>10)
t3: when(ev(v) [y:=v] if (x>0)
```

RIPN_TR.TXT FILE

```
FACTS: see
VARSINT: x
VARSREAL:
DISCRETE: act1(); act0()
DURATIVE: act2(INT)
TIMERS:
INIT: x:=0

<TR>
x==3 -> act2(x); _send("ev", 12) []
True -> act0(); act1() [remember(see)]
```

LOG_EV.TXT FILE

```
[LOG STARTED] 2025-05-14 09:33:39.415
[2025-05-14 09:33:42.531] Event added: ev
```

Active Events:
    ev [12.0] expires in 9223370289646753275 ms
Unattended Events:

[2025-05-14 09:33:43.595] Event consumed: ev with parameters [12.0]
Active Events:
Unattended Events:

[2025-05-14 09:33:48.552] Event added: ev
Active Events:
    ev [12.0] expires in 9223370289646747255 ms
Unattended Events:

PETRI NET LOG IN LOG_PN.TXT FILE

[2025-05-14 09:33:39.429] Current state of the Petri Net:
p1: ●
p2: ○
p3: ○
p4: ○
🚫 Transition 't4' blocked by condition: see && y>10
[2025-05-14 09:33:39.584] Current state of the Petri Net:
p1: ○
p2: ●
p3: ○
p4: ○
🚫 Transition 't4' blocked by condition: see && y>10
[2025-05-14 09:33:41.587] Current state of the Petri Net:
p1: ○
p2: ○
p3: ●
p4: ○
🚫 Transition 't4' blocked by condition: see && y>10
[2025-05-14 09:33:43.596] Current state of the Petri Net:
p1: ●
p2: ○
p3: ○
p4: ○
[2025-05-14 09:33:45.599] Current state of the Petri Net:
p1: ○
p2: ●
p3: ○
p4: ○
[2025-05-14 09:33:47.601] Current state of the Petri Net:
p1: ○
p2: ○
p3: ●
p4: ○
[2025-05-14 09:33:49.604] Current state of the Petri Net:
p1: ○
p2: ○
p3: ○
p4: ●
⏸ No fireable transitions at this moment. Waiting for changes...

TR PROGRAM LOG IN LOG_TR.TXT FILE

[2025-05-14 09:49:34.696] Initiating TR program...
[2025-05-14 09:49:34.710] 🔄 Executing rule with condition: True
[2025-05-14 09:49:34.710] ▶ Executing discrete action: act0 with parameters: []

[2025-05-14 09:49:34.715] ⏩ Executing discrete action: act1 with parameters: []
[2025-05-14 09:49:37.723] 🔄 Executing rule with condition: x==3
[2025-05-14 09:49:37.727] 📮 Event sent to environment: ev[12.0]
[2025-05-14 09:49:37.728] ⏳ Starting durative action: act2(x)
[2025-05-14 09:49:39.737] ✅ Stopping durative action: act2(x)
[2025-05-14 09:49:39.738] 🔄 Executing rule with condition: True
[2025-05-14 09:49:39.738] ⏩ Executing discrete action: act0 with parameters: []
[2025-05-14 09:49:39.738] ⏩ Executing discrete action: act1 with parameters: []
[2025-05-14 09:49:43.748] 🔄 Executing rule with condition: x==3
[2025-05-14 09:49:43.749] 📮 Event sent to environment: ev[12.0]
[2025-05-14 09:49:43.749] ⏳ Starting durative action: act2(x)

BELIEFSTORE LOG IN LOG_BS.TXT FILE

[2025-05-14 09:49:34.627]
🔷 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {}
  Integer variables: {v=0, x=0, y=0}
  Real variables: {}

[2025-05-14 09:49:34.785]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {v=0, x=2, y=0}
  Real variables: {}

[2025-05-14 09:49:36.791]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {v=0, x=3, y=0}
  Real variables: {}

[2025-05-14 09:49:38.796]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {v=12, x=1, y=12}
  Real variables: {}

[2025-05-14 09:49:40.797]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {v=12, x=2, y=12}
  Real variables: {}

[2025-05-14 09:49:42.799]
🔷 Current BeliefStore state:
  Active facts without parameters: [see]
  Active facts with parameters: {}
  Integer variables: {v=12, x=3, y=12}
  Real variables: {}

EXAMPLE OF RIPN SPECIFICATION "Irrigation System", given only by a Petri net

This Petri net models the behavior of an automated irrigation system that monitors soil moisture and controls a water valve accordingly. The system uses timers, facts, integer variables, and both discrete and durative actions to handle the irrigation cycle.

Initial Setup: The system starts in the idle state. A timer tCheck is initialized to periodically trigger a dryness check.
Idle Phase (idle): Starts timer tCheck with a duration of 10 seconds.
Timer Completion (tCheckEnd): When tCheck.end becomes true, the system forgets this timer expiration and returns to idle.
Dryness Check (tCheckDryness): When the dryDetected event occurs, the system records the fact dryDetected, Forgets any existing moist(...) fact, moves to the checking place.
Checking Phase (checking): Executes notifyStart() to signal the start of irrigation, waterValve(1) to open the valve with intensity 1.
Start Watering (tStartWatering): If moistureLevel < threshold (initially 0 < 2), transitions to watering.
Watering Phase (watering): On each activation, increments moistureLevel by 1.
Stop Conditions:
      By moisture: If moistureLevel >= threshold, transition tStopByMoisture fires.
   By user: If the event stopButton is detected, transition tStopByManual fires.
Moisture Detected (moistDetected): Executes notifyStop(moistureLevel), Forgets dryDetected, Remembers moist(moistureLevel). It also inhibits reactivation of tStartWatering via an inhibitor arc.
Send Alert (tSendAlert): Transitions the system back to idle and marks alertSent.

Additional Details:
- The timer tCheck is responsible for initiating periodic checks.
- The system uses events (dryDetected, stopButton) and shared facts (dryDetected, moist(...)) to coordinate behavior.
- Variables like moistureLevel and threshold control when to start or stop watering.
- The Petri net ensures deterministic flow and responsiveness to both environmental conditions and user input.
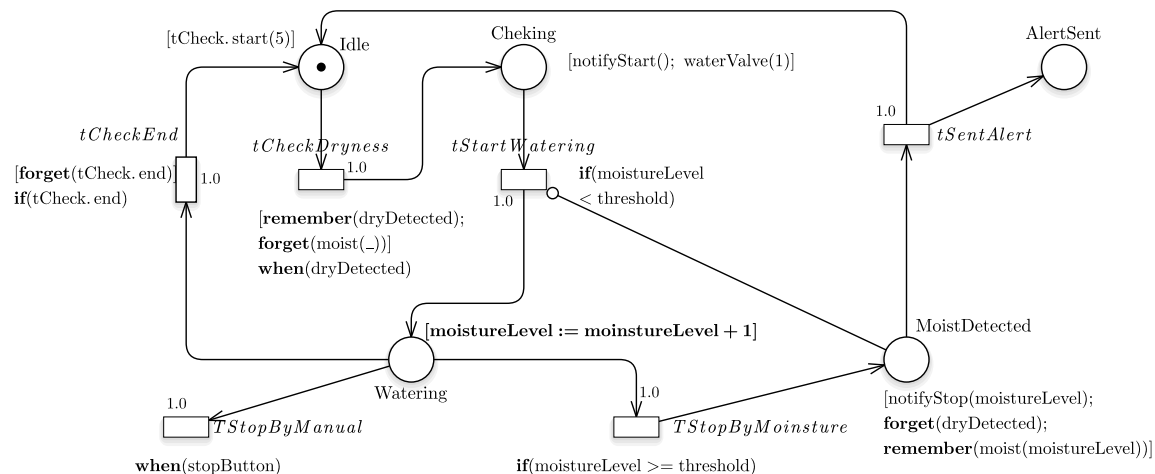
FACTS: dryDetected; moist(INT)
VARSINT: moistureLevel; threshold
VARSREAL: temperature
INIT: moistureLevel := 0; threshold := 2
DISCRETE: notifyStart(); notifyStop(INT)
DURATIVE: waterValve(INT)
TIMERS: tCheck
PLACES: idle; checking; watering; moistDetected; alertSent
TRANSITIONS: tCheckDryness; tStartWatering; tStopByMoisture; tStopByManual; tSendAlert; tCheckEnd;

ARCS: tCheckEnd->idle ; idle->tCheckDryness; watering->tCheckEnd; tCheckDryness->checking;
ARCS: checking->tStartWatering; tStartWatering->watering; watering->tStopByMoisture; watering->tStopByManual
ARCS: tStopByMoisture->moistDetected; moistDetected->tSendAlert; tSendAlert->idle; tSendAlert->alertSent; moistDetected -

o> tStartWatering

INITMARKING:(1,0,0,0,0)
EVENTS: dryDetected(0); stopButton(10)

<PN>
idle: [tCheck.start(10)]
tCheckEnd: [forget(tCheck.end)] if(tCheck.end)
tCheckDryness: when (dryDetected) [remember(dryDetected); forget(moist(_))]
checking: [notifyStart(); waterValve(1)]
tStartWatering: [] if (moistureLevel < threshold)
watering: [moistureLevel := moistureLevel + 1]
tStopByMoisture: [] if (moistureLevel >= threshold)
tStopByManual: when (stopButton) []
moistDetected: [notifyStop(moistureLevel); forget(dryDetected); remember(moist(moistureLevel))]

TRACE RESULTING OF THE EXECUTION WITH THREE "DRY_DETECTED" EVENT INSTANCES INJECTED TO THE SYSTEM

[2025-05-14 12:16:58.426] Current state of the Petri Net:
alertSent: ○
checking: ○
watering: ○
idle: ●
moistDetected: ○
[2025-05-14 12:16:58.427]
🔷 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {}
  Integer variables: {moistureLevel=0, threshold=2}
  Real variables: {temperature=0.0, }

[2025-05-14 12:16:59.453] ⏳ Timer started: tCheck for 10 seconds
🚫 Transition 'tCheckEnd' blocked by condition: tCheck.end
🚫 Transition 'tStopByMoisture' blocked by condition: moistureLevel >= threshold
⏸️ No non-immediate transitions enabled at this time. Waiting...
[2025-05-14 12:17:07.486] Observer: Executing discrete action: notifyStart with parameters: []
[2025-05-14 12:17:07.487] Observer: Starting durative action: waterValve with parameters: [1.0]
[2025-05-14 12:17:07.489] Current state of the Petri Net:
alertSent: ○
checking: ●
watering: ○
idle: ○
moistDetected: ○
[2025-05-14 12:17:07.489]
🔷 Current BeliefStore state:
  Active facts without parameters: [dryDetected]
  Active facts with parameters: {}
  Integer variables: {moistureLevel=0, threshold=2}
  Real variables: {temperature=0.0, }

[2025-05-14 12:17:09.491] ✅ Timer expired: tCheck_end activated
[2025-05-14 12:17:09.491] 🔴 Timer fully removed: tCheck
[2025-05-14 12:17:09.491] Observer: Executing discrete action: notifyStart with parameters: []
🚫 Transition 'tStopByMoisture' blocked by condition: moistureLevel >= threshold
[2025-05-14 12:17:09.494] Observer: Stopping durative action: waterValve
[2025-05-14 12:17:09.494] Current state of the Petri Net:
alertSent: ○
checking: ○

watering: ●
idle: ○
moistDetected: ○
[2025-05-14 12:17:09.494]
🔷 Current BeliefStore state:
Active facts without parameters: [tCheck_end, dryDetected]
Active facts with parameters: {}
Integer variables: {moistureLevel=1, threshold=2}
Real variables: {temperature=0.0, }

🚫 Transition 'tStopByMoisture' blocked by condition: moistureLevel >= threshold
🗑 Fact removed: tCheck_end
[2025-05-14 12:17:11.501] ⌛ Timer started: tCheck for 10 seconds
[2025-05-14 12:17:11.502] 🚫 Skipped actions in transition tCheckEnd (Condition not met: tCheck.end)
[2025-05-14 12:17:11.502] Current state of the Petri Net:
alertSent: ○
checking: ○
watering: ○
idle: ●
moistDetected: ○
[2025-05-14 12:17:11.502]
🔷 Current BeliefStore state:
Active facts without parameters: [dryDetected]
Active facts with parameters: {}
Integer variables: {moistureLevel=1, threshold=2}
Real variables: {temperature=0.0, }

[2025-05-14 12:17:13.505] ⌛ Timer started: tCheck for 10 seconds
🚫 Transition 'tCheckEnd' blocked by condition: tCheck.end
🚫 Transition 'tStopByMoisture' blocked by condition: moistureLevel >= threshold
⏸ No non-immediate transitions enabled at this time. Waiting...
[2025-05-14 12:17:17.515] Observer: Executing discrete action: notifyStart with parameters: []
[2025-05-14 12:17:17.515] Observer: Starting durative action: waterValve with parameters: [1.0]
[2025-05-14 12:17:17.515] Current state of the Petri Net:
alertSent: ○
checking: ●
watering: ○
idle: ○
moistDetected: ○
[2025-05-14 12:17:19.518] Observer: Executing discrete action: notifyStart with parameters: []
🚫 Transition 'tCheckEnd' blocked by condition: tCheck.end
🚫 Transition 'tStopByMoisture' blocked by condition: moistureLevel >= threshold
[2025-05-14 12:17:19.520] 🚫 Skipped actions in transition tStartWatering (Condition not met: moistureLevel < threshold)
[2025-05-14 12:17:19.520] Observer: Stopping durative action: waterValve
[2025-05-14 12:17:19.520] Current state of the Petri Net:
alertSent: ○
checking: ○
watering: ●
idle: ○
moistDetected: ○
[2025-05-14 12:17:19.520]
🔷 Current BeliefStore state:
Active facts without parameters: [dryDetected]
Active facts with parameters: {}
Integer variables: {moistureLevel=2, threshold=2}
Real variables: {temperature=0.0, }

🚫 Transition 'tCheckEnd' blocked by condition: tCheck.end

🚫 Transition 'tStartWatering' blocked by condition: moistureLevel < threshold

[2025-05-14 12:17:21.522] Observer: Executing discrete action: notifyStop with parameters: [2.0]

🗑 Fact removed: dryDetected

[2025-05-14 12:17:21.542] Current state of the Petri Net:

alertSent: ○

checking: ○

watering: ○

idle: ○

moistDetected: ●

[2025-05-14 12:17:21.542]

🔷 Current BeliefStore state:

  Active facts without parameters: []

  Active facts with parameters: {moist=[[2]], }

  Integer variables: {moistureLevel=2, threshold=2}

  Real variables: {temperature=0.0, }

[2025-05-14 12:17:23.547] ✅ Timer expired: tCheck_end activated

[2025-05-14 12:17:23.547] 🔴 Timer fully removed: tCheck

[2025-05-14 12:17:23.547] Observer: Executing discrete action: notifyStop with parameters: [2.0]

🚫 Transition 'tStartWatering' blocked by condition: moistureLevel < threshold

🗑 Fact removed: tCheck_end

[2025-05-14 12:17:23.549] ⏳ Timer started: tCheck for 10 seconds

[2025-05-14 12:17:23.549] Current state of the Petri Net:

alertSent: ●

checking: ○

watering: ○

idle: ●

moistDetected: ○

🚫 Transition 'tCheckEnd' blocked by condition: tCheck.end

🚫 Transition 'tStartWatering' blocked by condition: moistureLevel < threshold

⏸ No non-immediate transitions enabled at this time. Waiting...

[2025-05-14 12:17:27.559] ⏳ Timer started: tCheck for 10 seconds

[2025-05-14 12:17:29.565] Observer: Executing discrete action: notifyStart with parameters: []

[2025-05-14 12:17:29.566] Observer: Starting durative action: waterValve with parameters: [1.0]

[2025-05-14 12:17:29.566] Current state of the Petri Net:

alertSent: ●

checking: ●

watering: ○

idle: ○

moistDetected: ○

[2025-05-14 12:17:29.566]

🔷 Current BeliefStore state:

  Active facts without parameters: [dryDetected]

  Active facts with parameters: {}

  Integer variables: {moistureLevel=2, threshold=2}

  Real variables: {temperature=0.0, }

[2025-05-14 12:17:31.569] Observer: Executing discrete action: notifyStart with parameters: []

🚫 Transition 'tCheckEnd' blocked by condition: tCheck.end

🚫 Transition 'tStartWatering' blocked by condition: moistureLevel < threshold

⏸ No fireable transitions at this moment. Waiting for changes...

[2025-05-14 12:17:37.578] ✅ Timer expired: tCheck_end activated

[2025-05-14 12:17:37.578] 🔴 Timer fully removed: tCheck

EXAMPLE OF RIPN SPECIFICATION "Irrigation System", given only by a TR program

FACTS:
PERCEPTS: fin; reset
VARSINT: x
VARSREAL:
DISCRETE: do(); nil()
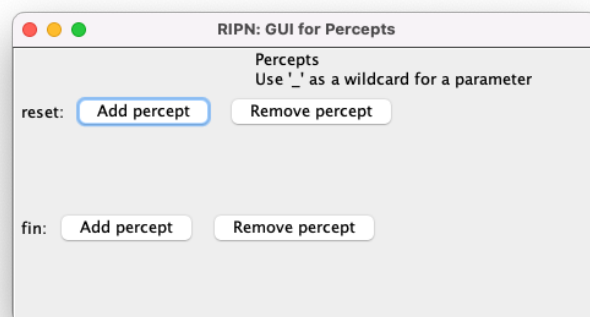DURATIVE: do2()
TIMERS: timer
INIT: x:=0

```
<TR>
reset -> [forget(timer.end); x:=0]
fin -> nil()
timer.end -> do2()
x==1 -> do()
True -> timer.start(5) [x:=1]
```


This TR program defines a simple behavior controller that uses a timer and two percepts (fin, reset) to manage a sequence of actions. Here's what each part does:

Declarations
- **FACTS:** No facts are declared.
- **PERCEPTS:**
  - fin: a percept without parameters.
  - reset: another percept without parameters.
- **VARSINT:**
  - x: an integer variable used to control behavior.
- **VARSREAL:**
  - None.
- **DISCRETE ACTIONS:**
  - do(): a discrete action.
  - nil(): a no-op or placeholder action.
- **DURATIVE ACTIONS:**
  - do2(): a durative action (executed while conditions remain true).
- **TIMERS:**
  - timer: a timer that can be started, stopped, etc.
- **INIT:**
  - x is initialized to 0.

Rules
The rules define how the system reacts to percepts and the passage of time:
1. **reset -> [forget(timer.end); x:=0]**
   When the percept reset is active, the system forgets that the timer has ended (if it had), and resets x to 0.
2. **fin -> nil()**

If the percept fin is active, the system does nothing (a placeholder action).
3. **timer.end -> do2()**
   When the timer finishes (after 5 units of time), the durative action do2() is started.
4. **x==1 -> do()**
   If x is equal to 1, the discrete action do() is executed.
5. **True -> timer.start(5) [x:=1]**
   This is the default rule. If no other rule matches, this one is always true. It starts the timer with a duration of 5, and sets x to 1.

Behavior Summary
- Initially, x = 0, and the timer is not running.
- Since no conditions are met, the last rule (True) triggers: it starts the timer and sets x to 1.
- Now that x == 1, the rule x==1 -> do() becomes active and executes the do() action.
- After the timer ends, the rule timer.end -> do2() becomes active and starts the durative action do2().
- If at any point the reset percept appears, the timer end is forgotten and x is reset to 0, effectively restarting the cycle.
- If the fin percept is observed, nothing happens due to the nil() action, but this could be a placeholder for future extension.

A possible trace of the system in execution is the following:

ℹ️ Initialized integer variable 'x' to 0
Initiating tr program...
[2025-06-02 11:06:37.155]
🔷 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {}
  Integer variables: {x=0}
  Real variables: {}

[2025-06-02 11:06:37.184] 🔄 Executing rule with condition: True
[2025-06-02 11:06:37.185] 🛠 Extracted timer command: start for timer: timer
[2025-06-02 11:06:37.185] ⏳ Timer started: timer for 5 seconds
[2025-06-02 11:06:37.290]
🔷 Current BeliefStore state:
  Active facts without parameters: []
  Active facts with parameters: {}
  Integer variables: {x=1}
  Real variables: {}

[2025-06-02 11:06:37.291] 🔄 Executing rule with condition: x==1
[2025-06-02 11:06:37.291] ⏩ Executing discrete action: do with parameters: []
[2025-06-02 11:06:37.302] Observer: Executing discrete action: do with parameters: []
[2025-06-02 11:06:42.237] ✅ Timer expired: timer_end activated
[2025-06-02 11:06:42.237] 🔴 Timer fully removed: timer
[2025-06-02 11:06:42.339]
🔷 Current BeliefStore state:
  Active facts without parameters: [timer_end]
  Active facts with parameters: {}
  Integer variables: {x=1}
  Real variables: {}

[2025-06-02 11:06:42.339] 🔄 Executing rule with condition: timer.end
[2025-06-02 11:06:42.340] ⏳ Starting durative action: do2()
[2025-06-02 11:06:42.340] Observer: Starting durative action: do2() with parameters: []
[2025-06-02 11:07:54.468]
🔷 Current BeliefStore state:
  Active facts without parameters: [timer_end, fin]

Active facts with parameters: {}
Integer variables: {x=1}
Real variables: {}

[2025-06-02 11:07:54.468] ✅ Stopping durative action: do2()
[2025-06-02 11:07:54.469] Observer: Stopping durative action: do2()
[2025-06-02 11:07:54.469] 🔄 Executing rule with condition: fin
[2025-06-02 11:07:54.469] ⏩ Executing discrete action: nil with parameters: []
[2025-06-02 11:07:54.469] Observer: Executing discrete action: nil with parameters: []
[2025-06-02 11:08:01.147]
🔷 Current BeliefStore state:
Active facts without parameters: [timer_end, reset, fin]
Active facts with parameters: {}
Integer variables: {x=1}
Real variables: {}

[2025-06-02 11:08:01.147] 🔄 Executing rule with condition: reset
🗑 Fact removed: timer_end
[2025-06-02 11:08:01.249]
🔷 Current BeliefStore state:
Active facts without parameters: [reset, fin]
Active facts with parameters: {}
Integer variables: {x=0}
Real variables: {}

🗑 Fact removed: reset
[2025-06-02 11:08:07.610]
🔷 Current BeliefStore state:
Active facts without parameters: [fin]
Active facts with parameters: {}
Integer variables: {x=0}
Real variables: {}

[2025-06-02 11:08:07.611] 🔄 Executing rule with condition: fin
[2025-06-02 11:08:07.611] ⏩ Executing discrete action: nil with parameters: []
[2025-06-02 11:08:07.611] Observer: Executing discrete action: nil with parameters: []
🗑 Fact removed: fin
[2025-06-02 11:08:09.479]
🔷 Current BeliefStore state:
Active facts without parameters: []
Active facts with parameters: {}
Integer variables: {x=0}
Real variables: {}

[2025-06-02 11:08:09.480] 🔄 Executing rule with condition: True
[2025-06-02 11:08:09.480] 🔧 Extracted timer command: start for timer: timer
[2025-06-02 11:08:09.480] ⏳ Timer started: timer for 5 seconds
[2025-06-02 11:08:09.584]
🔷 Current BeliefStore state:
Active facts without parameters: []
Active facts with parameters: {}
Integer variables: {x=1}
Real variables: {}

[2025-06-02 11:08:09.585] 🔄 Executing rule with condition: x==1
[2025-06-02 11:08:09.585] ⏩ Executing discrete action: do with parameters: []
[2025-06-02 11:08:09.585] Observer: Executing discrete action: do with parameters: []
[2025-06-02 11:08:14.575] ✅ Timer expired: timer_end activated

[2025-06-02 11:08:14.575] 🛑 Timer fully removed: timer
[2025-06-02 11:08:14.675]
🔷 Current BeliefStore state:
   Active facts without parameters: [timer_end]
   Active facts with parameters: {}
   Integer variables: {x=1}
   Real variables: {}

[2025-06-02 11:08:14.676] 🔄 Executing rule with condition: timer.end
[2025-06-02 11:08:14.676] ⏳ Starting durative action: do2()
[2025-06-02 11:08:14.676] Observer: Starting durative action: do2() with parameters: []

EXAMPLE OF RIPN SPECIFICATION, both Petri net and TR program

This is an interesting example because it includes a bit of all of RIPN's functionality. From the TR program, events are sent to the environment to be captured by the Petri net (_send). It has a timer, executes durative and discrete actions, handles percepts received from the GUI environment, adds a fact to the BeliefStore, and modifies a variable. From the Petri net, facts are removed from the BeliefStore. Up to three events are served (received from the TR program or from the environment), the first two of which have no expiration date and the third (ev3) expires after 10 seconds. The net associates place p3 with the execution of a durative action, so as long as the place is marked, the action will be executed. However, place p4 has a discrete action associated with it, so only transitioning from the unmarked state to the marked state will trigger that action into the environment.

PN
FACTS: fact1
VARSINT: x
VARSREAL:
INIT:
DISCRETE: adi2()
DURATIVE: adu2()
TIMERS: timer1
PLACES: p0; p1; p2; p3; p4
TRANSITIONS: t1; t2; t3; t4
ARCS: p0->t1; t1->p1; p1->t2; t2->p2; p2->t3; t3->p0; t3->p3; p3->t4; t4->p4; p4->t3

INITMARKING: (1,0,0,1,0)
EVENTS: ev1(0); ev2(0); ev3(10)

<PN>
p0: [x:=1; forget(timer1.end)]
t1: when(ev1) [] if(x==1)
p1: [x:=2]
t2: when(ev2) []
p2: [x:=3]
t3: when(ev3) [forget(fact1)]
p3: [adu2()]
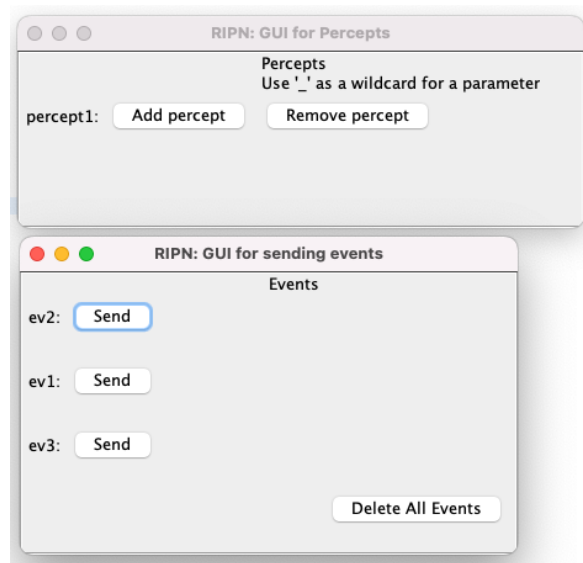t4: [] if(fact1)
p4: [adi(2)]

TR
FACTS: fact1
PERCEPTS: percept1
VARSINT: x
VARSREAL:
DISCRETE: nil(); adi(INT, INT)
DURATIVE: adu(INT)
TIMERS: timer1
INIT:

<TR>
timer1.end -> nil()

percept1 -> adi(1,2)
x==3 && fact1-> _send("ev3")
x==1 -> adi(3,4) [remember(fact1)]
True -> adu(5); timer1.start(20) [x:=1]

RIPN: GUI for Percepts

Percepts
Use '_' as a wildcard for a parameter

percept1: [Add percept] [Remove percept]

RIPN: GUI for sending events

Events

ev2: [Send]

ev1: [Send]

ev3: [Send]

[Delete All Events]

Console ✕

MainUnified (4) [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java (11 jun 2025, 10:32:34

```
...started Petri net.
...started TR program.

[2025-06-11 10:32:36.422] Observer: Starting durative action: adu2 with parameters: []
[2025-06-11 10:32:37.733] Observer: Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:32:37.787] Observer: Stopping durative action: adu2
[2025-06-11 10:33:51.932] Observer: Starting durative action: adu(5) with parameters: [5.0]
[2025-06-11 10:33:52.935] Observer: Stopping durative action: adu(5)
[2025-06-11 10:33:52.936] Observer: Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:33:55.886] Observer: Starting durative action: adu2 with parameters: []
[2025-06-11 10:33:55.948] Observer: Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:33:56.953] Observer: Executing discrete action: adi with parameters: [1.0, 2.0]
[2025-06-11 10:33:57.888] Observer: Stopping durative action: adu2
[2025-06-11 10:34:11.903] Observer: Starting durative action: adu2 with parameters: []
[2025-06-11 10:34:11.994] Observer: Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:34:12.998] Observer: Executing discrete action: nil with parameters: []
[2025-06-11 10:34:13.905] Observer: Stopping durative action: adu2
[2025-06-11 10:34:14.002] Observer: Executing discrete action: adi with parameters: [3.0, 4.0]
```

TR PROGRAM TRACE

[2025-06-11 10:32:37.629] Initiating TR program...
[2025-06-11 10:32:37.716] 🔄 Executing rule with condition: x==1
[2025-06-11 10:32:37.721] ⏩ Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:33:51.931] 🔄 Executing rule with condition: True
[2025-06-11 10:33:51.932] 🔧 Extracted timer command: start for timer: timer1
[2025-06-11 10:33:51.932] ⏳ Starting durative action: adu(5)
[2025-06-11 10:33:52.935] ✅ Stopping durative action: adu(5)
[2025-06-11 10:33:52.936] 🔄 Executing rule with condition: x==1
[2025-06-11 10:33:52.936] ⏩ Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:33:53.942] 🔄 Executing rule with condition: x==3 && fact1
[2025-06-11 10:33:53.945] 📨 Event sent to environment: ev3[]
[2025-06-11 10:33:55.947] 🔄 Executing rule with condition: x==1
[2025-06-11 10:33:55.948] ⏩ Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:33:56.952] 🔄 Executing rule with condition: percept1
[2025-06-11 10:33:56.952] ⏩ Executing discrete action: adi with parameters: [1.0, 2.0]
[2025-06-11 10:34:09.984] 🔄 Executing rule with condition: x==3 && fact1
[2025-06-11 10:34:09.985] 📨 Event sent to environment: ev3[]

[2025-06-11 10:34:11.993] 🔄 Executing rule with condition: x==1
[2025-06-11 10:34:11.994] ⏯ Executing discrete action: adi with parameters: [3.0, 4.0]
[2025-06-11 10:34:12.998] 🔄 Executing rule with condition: timer1.end
[2025-06-11 10:34:12.998] ⏯ Executing discrete action: nil with parameters: []
[2025-06-11 10:34:14.002] 🔄 Executing rule with condition: x==1
[2025-06-11 10:34:14.002] ⏯ Executing discrete action: adi with parameters: [3.0, 4.0]

PETRI NET TRACE

[2025-06-11 10:32:36.422] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
p3: ●
p4: ○
[2025-06-11 10:32:37.787] 🔥 Transition fired: t4
[2025-06-11 10:32:37.787] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
p3: ○
p4: ●
⏸ No non-immediate transitions enabled at this time. Waiting...
[2025-06-11 10:33:51.881] 🚫 Skipped actions in transition t1 (Condition not met: x==1)
[2025-06-11 10:33:51.881] 🔥 Transition fired: t1
[2025-06-11 10:33:51.881] Current state of the Petri Net:
p0: ○
p1: ●
p2: ○
p3: ○
p4: ●
🚫 Transition 't1' blocked by condition: x==1
[2025-06-11 10:33:53.884] 🔥 Transition fired: t2
[2025-06-11 10:33:53.884] Current state of the Petri Net:
p0: ○
p1: ○
p2: ●
p3: ○
p4: ●
🚫 Transition 't1' blocked by condition: x==1
[2025-06-11 10:33:55.886] 🔥 Transition fired: t3
[2025-06-11 10:33:55.886] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
p3: ●
p4: ○
[2025-06-11 10:33:57.887] 🔥 Transition fired: t4
[2025-06-11 10:33:57.888] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
p3: ○
p4: ●
⏸ No non-immediate transitions enabled at this time. Waiting...
[2025-06-11 10:34:03.895] 🚫 Skipped actions in transition t1 (Condition not met: x==1)
[2025-06-11 10:34:03.895] 🔥 Transition fired: t1

[2025-06-11 10:34:03.895] Current state of the Petri Net:
p0: ○
p1: ●
p2: ○
p3: ○
p4: ●
🚫 Transition 't1' blocked by condition: x==1
[2025-06-11 10:34:05.897] 🔥 Transition fired: t2
[2025-06-11 10:34:05.897] Current state of the Petri Net:
p0: ○
p1: ○
p2: ●
p3: ○
p4: ●
🚫 Transition 't1' blocked by condition: x==1
⏸️ No non-immediate transitions enabled at this time. Waiting...
[2025-06-11 10:34:11.903] 🔥 Transition fired: t3
[2025-06-11 10:34:11.903] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
p3: ●
p4: ○
[2025-06-11 10:34:13.905] 🔥 Transition fired: t4
[2025-06-11 10:34:13.905] Current state of the Petri Net:
p0: ●
p1: ○
p2: ○
p3: ○
p4: ●
⏸️ No non-immediate transitions enabled at this time. Waiting...

EVENTS TRACE

[LOG STARTED] 2025-06-11 10:32:36.311
[2025-06-11 10:33:44.442] Event added: ev2
Active Events:
    ev2 [] expires in 9223370287223951365 ms
Unattended Events:

[2025-06-11 10:33:50.076] Event added: ev1
Active Events:
    ev2 [] expires in 9223370287223945731 ms
    ev1 [] expires in 9223370287223945731 ms
Unattended Events:

[2025-06-11 10:33:51.880] Event consumed: ev1 with parameters []
Active Events:
    ev2 [] expires in 9223370287223943927 ms
Unattended Events:

[2025-06-11 10:33:53.883] Event consumed: ev2 with parameters []
Active Events:
Unattended Events:

[2025-06-11 10:33:53.944] Event added: ev3
Active Events:
    ev3 [] expires in 9999 ms
Unattended Events:

[2025-06-11 10:33:55.885] Event consumed: ev3 with parameters []
Active Events:
Unattended Events:

[2025-06-11 10:34:02.073] Event added: ev1
Active Events:
 ev1 [] expires in 9223370287223933734 ms
Unattended Events:

[2025-06-11 10:34:03.894] Event consumed: ev1 with parameters []
Active Events:
Unattended Events:

[2025-06-11 10:34:04.609] Event added: ev2
Active Events:
 ev2 [] expires in 9223370287223931198 ms
Unattended Events:

[2025-06-11 10:34:05.897] Event consumed: ev2 with parameters []
Active Events:
Unattended Events:

[2025-06-11 10:34:09.985] Event added: ev3
Active Events:
 ev3 [] expires in 9999 ms
Unattended Events:

[2025-06-11 10:34:11.902] Event consumed: ev3 with parameters []
Active Events:
Unattended Events:

BELIEFSTORE TRACE

ℹ️ Initialized integer variable 'x' to 0
[2025-06-11 10:32:36.422]
🔷 Current BeliefStore state:
 Active facts without parameters: []
 Active facts with parameters: {}
 Integer variables: {x=0}
 Real variables: {}

[2025-06-11 10:32:37.788]
🔷 Current BeliefStore state:
 Active facts without parameters: [fact1]
 Active facts with parameters: {}
 Integer variables: {x=1}
 Real variables: {}

[2025-06-11 10:33:51.881]
🔷 Current BeliefStore state:
 Active facts without parameters: [fact1]
 Active facts with parameters: {}
 Integer variables: {x=2}
 Real variables: {}

[2025-06-11 10:33:51.932] ⏳ Timer started: timer1 for 20 seconds
[2025-06-11 10:33:52.935]
🔷 Current BeliefStore state:

Active facts without parameters: [fact1]
Active facts with parameters: {}
Integer variables: {x=1}
Real variables: {}

[2025-06-11 10:33:53.884]
◆ Current BeliefStore state:
Active facts without parameters: [fact1]
Active facts with parameters: {}
Integer variables: {x=3}
Real variables: {}

🗑 Fact removed: fact1
[2025-06-11 10:33:55.886]
◆ Current BeliefStore state:
Active facts without parameters: []
Active facts with parameters: {}
Integer variables: {x=1}
Real variables: {}

[2025-06-11 10:33:56.952]
◆ Current BeliefStore state:
Active facts without parameters: [percept1, fact1]
Active facts with parameters: {}
Integer variables: {x=1}
Real variables: {}

[2025-06-11 10:34:03.895]
◆ Current BeliefStore state:
Active facts without parameters: [percept1, fact1]
Active facts with parameters: {}
Integer variables: {x=2}
Real variables: {}

[2025-06-11 10:34:05.897]
◆ Current BeliefStore state:
Active facts without parameters: [percept1, fact1]
Active facts with parameters: {}
Integer variables: {x=3}
Real variables: {}

🗑 Fact removed: percept1
[2025-06-11 10:34:09.983]
◆ Current BeliefStore state:
Active facts without parameters: [fact1]
Active facts with parameters: {}
Integer variables: {x=3}
Real variables: {}

🗑 Fact removed: fact1
[2025-06-11 10:34:11.903]
◆ Current BeliefStore state:
Active facts without parameters: []
Active facts with parameters: {}
Integer variables: {x=1}
Real variables: {}

[2025-06-11 10:34:11.994] ✅ Timer expired: timer1_end activated
[2025-06-11 10:34:11.994] 🛑 Timer fully removed: timer1

[2025-06-11 10:34:12.997]
◆ Current BeliefStore state:
   Active facts without parameters: [timer1_end, fact1]
   Active facts with parameters: {}
   Integer variables: {x=1}
   Real variables: {}

🗑 Fact removed: timer1_end
[2025-06-11 10:34:13.905]
◆ Current BeliefStore state:
   Active facts without parameters: [fact1]
   Active facts with parameters: {}
   Integer variables: {x=1}
   Real variables: {}

BNF OF PETRI NET FILES

<program> ::= <declarations> <net-definition>

<declarations> ::=
    ["FACTS:" <fact-list>]
    ["VARSINT:" <varint-list>]
    ["VARSREAL:" <varreal-list>]
    ["INIT:" <init-assignments>]
    ["DISCRETE:" <action-list>]
    ["DURATIVE:" <action-list>]
    ["TIMERS:" <timer-list>]
    ["EVENTS:" <event-declaration-list>]
    "PLACES:" <place-list>
    "TRANSITIONS:" <transition-list>
    "ARCS:" <arc-list>
    "INITMARKING:" "(" <marking-list> ")"

<event-declaration-list> ::= <event-declaration> (";" <event-declaration>)*

<event-declaration> ::= <identifier> "(" <event-duration> ["," <event-param-types>]? ")"
<event-duration> ::= <int>
<event-param-types> ::= <event-param-type> ("," <event-param-type>)*
<event-param-type> ::= "INT" | "REAL"

<net-definition> ::= "<PN>" <pn-line>*

<pn-line> ::= <identifier> ":" <block-content>

<block-content> ::= "[" <actions> "]" ["if" "(" <condition> ")"] ["when" "(" <event-reference> ")"]
        | "[]" ["if" "(" <condition> ")"] ["when" "(" <event-reference> ")"]

<actions> ::= <action> (";" <action>)*
<action> ::= <assignment> | <discrete-action-call> | <durative-action-call> | <timer-command> | <fact-command>

<assignment> ::= <identifier> ":=" <expression>
<fact-command> ::= "remember(" <fact-instance> ")" | "forget(" <fact-instance> ")"
<fact-instance> ::= <identifier> ["(" <expression-list> ")"]
<discrete-action-call> ::= <identifier> "(" [<expression-list>] ")"
<durative-action-call> ::= <identifier> "(" [<expression-list>] ")"
<timer-command> ::= <identifier> "." ("start" | "stop" | "pause" | "continue") "(" [<expression>] ")"

<event-reference> ::= <identifier> "(" [<identifier-list>] ")"
<identifier-list> ::= <identifier> ("," <identifier>)*

<expression-list> ::= <expression> ("," <expression>)*
<expression> ::= <term> (("+" | "-" | "*" | "/") <term>)*
<term> ::= <number> | <identifier> | "(" <expression> ")"

<condition> ::= <logical-expression>
<logical-expression> ::= <logical-term> (("||" | "&&") <logical-term>)*
<logical-term> ::= <expression> <comparison-op> <expression>
       | <fact-condition>

<comparison-op> ::= "==" | "!=" | ">" | ">=" | "<" | "<="

<fact-condition> ::= <identifier> "(" [<fact-param-list>] ")"
<fact-param-list> ::= <fact-param> ("," <fact-param>)*
<fact-param> ::= "out" <identifier> | <identifier> | "_" | <number> | <expression>

<fact-list> ::= <fact-decl> (";" <fact-decl>)*
<fact-decl> ::= <identifier> ["(" <param-type-list> ")"]

```
<varint-list> ::= <identifier> (";" <identifier>)*
<varreal-list> ::= <identifier> (";" <identifier>)*

<init-assignments> ::= <init-entry> (";" <init-entry>)*
<init-entry> ::= <assignment>
            | <fact-instance>
            | <fact-range>

<fact-range> ::= <identifier> "(" <int> ".." <int> ")"

<action-list> ::= <action-decl> (";" <action-decl>)*
<action-decl> ::= <identifier> "(" [ <param-type-list> ] ")"
<param-type-list> ::= <param-type> ("," <param-type>)*
<param-type> ::= "INT" | "REAL"

<timer-list> ::= <identifier> (";" <identifier>)*
<place-list> ::= <identifier> (";" <identifier>)*
<transition-list> ::= <identifier> (";" <identifier>)*
<arc-list> ::= <arc> (";" <arc>)*
<arc> ::= <place-or-transition> ("->" | "-o>") <place-or-transition>
<place-or-transition> ::= <identifier>

<marking-list> ::= <int> ("," <int>)*

<identifier> ::= [a-zA-Z_] [a-zA-Z0-9_]*
<number> ::= [0-9]+ ("." [0-9]+)?
<int> ::= [0-9]+
```

BNF OF TR PROGRAM FILES

```
<program> ::= <declarations> "<TR>" <rule-list>

<declarations> ::=
    [ "FACTS:" <fact-list> ]
    [ "PERCEPTS:" <fact-list> ]
    [ "VARSINT:" <varint-list> ]
    [ "VARSREAL:" <varreal-list> ]
    [ "DISCRETE:" <action-list> ]
    [ "DURATIVE:" <action-list> ]
    [ "TIMERS:" <timer-list> ]
    [ "INIT:" <init-assignments> ]

<fact-list> ::= <fact-decl> (";" <fact-decl>)*
<fact-decl> ::= <identifier> [ "(" <param-type-list> ")" ]

<varint-list> ::= <identifier> (";" <identifier>)*
<varreal-list> ::= <identifier> (";" <identifier>)*

<action-list> ::= <action-decl> (";" <action-decl>)*
<action-decl> ::= <identifier> "(" [ <param-type-list> ] ")"

<timer-list> ::= <identifier> (";" <identifier>)*

<param-type-list> ::= <param-type> ("," <param-type>)*
<param-type> ::= "INT" | "REAL"

<init-assignments> ::= <init-entry> (";" <init-entry>)*
<init-entry> ::= <assignment>
            | <fact-instance>
            | <fact-range>

<fact-range> ::= <identifier> "(" <int> ".." <int> ")"

<rule-list> ::= <rule>*
<rule> ::= <condition> "->" <actions> [ <updates> ]

<condition> ::= <logical-expression>

<logical-expression> ::= <logical-term> (("&&" | "||") <logical-term>)*
<logical-term> ::= <expression> <comparison-op> <expression>
            | <fact-instance>
            | <timer-check>

<comparison-op> ::= "==" | "!=" | ">" | ">=" | "<" | "<="

<fact-instance> ::= <identifier> "(" [ <fact-argument-list> ] ")"
<fact-argument-list> ::= <fact-arg> ("," <fact-arg>)*
<fact-arg> ::= "out" <identifier> | <expression> | "_"

<timer-check> ::= <identifier> ".end"

<actions> ::= <action-call> (";" <action-call>)*
<action-call> ::= <identifier> "(" [ <expression-list> ] ")"
            | <timer-name> "." <timer-op> "(" [ <expression> ] ")"
            | "_send" "(" <string-literal> [ "," <expression-list> ] ")"

<timer-op> ::= "start" | "stop" | "pause" | "continue"

<updates> ::= "[" <update-entry> (";" <update-entry>)* "]"
```

```
<update-entry> ::= <assignment>
          | "remember(" <fact-instance> ")"
          | "forget(" <fact-instance> ")"

<assignment> ::= <identifier> ":=" <expression>

<expression-list> ::= <expression> ("," <expression>)*
<expression> ::= <term> (("+" | "-" | "*" | "/") <term>)*
<term> ::= <identifier> | <number> | "(" <expression> ")"

<string-literal> ::= "\"" <identifier> "\""
<identifier> ::= [a-zA-Z_] [a-zA-Z0-9_]*
<number> ::= [0-9]+ ("." [0-9]+)?
<int> ::= [0-9]+
```

Work in progress...

This first version has been subjected to various tests to validate its operation, both for the Petri nets and the TR programs separately and as a whole (shared BeliefStore and sending events from the TR program to the environment to be handled by the Petri net).

Messages have been included in the log regarding possible syntactic errors in the specifications. However, it is possible that not all situations have been covered and the software will continue to be tested leading to new versions.

You can cite RIPN framework by using:
P. Sánchez (2025). RIPN: (v1.3). Zenodo. https://doi.org/10.5281/zenodo.15697539

If you have any comments or questions, please write to **pedro.sanchez@upct.es**, Universidad Politécnica de Cartagena, Spain.