Petri Net Plans Execution Framework

SAPIENZA
UNIVERSITÀ DI ROMA

Luca Iocchi
Dipartimento di Ingegneria Informatica
Automatica e Gestionale

Petri Net Plans

- High-level plan representation formalism based on Petri nets
- Explicit and formal representation of actions and conditions
- Execution Algorithm implemented and tested in many robotic applications
- Open-source release with support for different robots and development environments (ROS, Naoqi, …)

Petri Net Plans library

PNP library contains

- PNP execution engine
- PNP generation tools
- Bridges: ROS, Naoqi (Nao, Pepper)

pnp.dis.uniroma1.it

[Ziparo et al., JAAMAS 2011]

Plan representation in PNP

- Petri nets are exponentially more compact than other structures (e.g., transition graphs) and can thus efficiently represent several kinds of plans:
  - Linear plans
  - Contingent/conditional plans
  - Plans with loop
  - Policies
  - …
- PNP can be used as a general plan execution framework

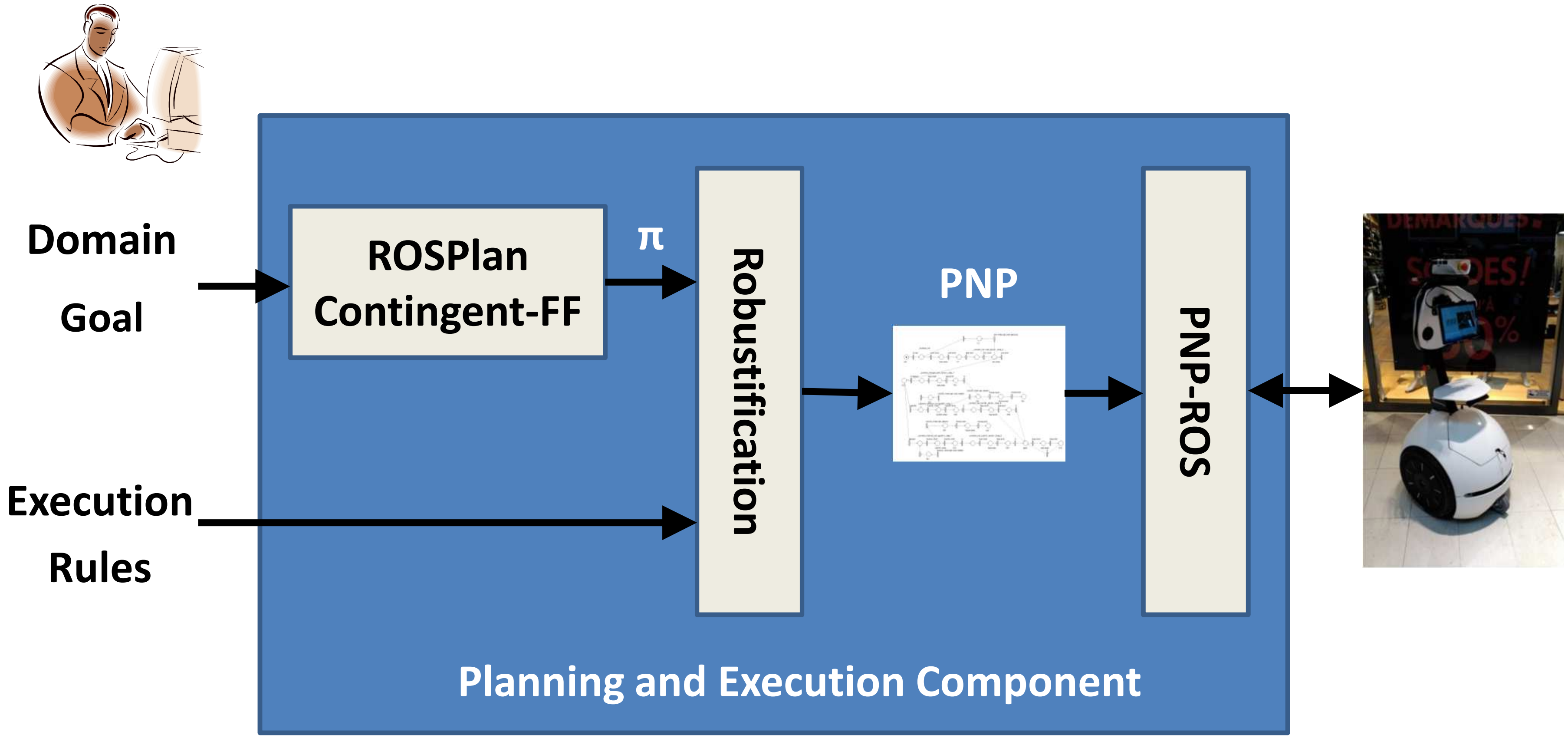
Plan traslation in PNP

- **PNPgen** is a library that translates a plan (the output of some planning system) in a PNP.
- **PNPgen** includes additional facilities to extend the generated PNP with constructs that are not available on the planning system (e.g., interrupt and recovery procedures).
- Plan formats supported:
  ROSPlan (linear/conditional), HATP, MDP policies

PNP ROS

- **PNP-ROS** is a bridge for executing PNPs in a ROS-based system.
- **PNP-ROS** uses the ROS actionlib protocol to control the execution of the actions and ROS topics and parameters to access the robot's knowledge.

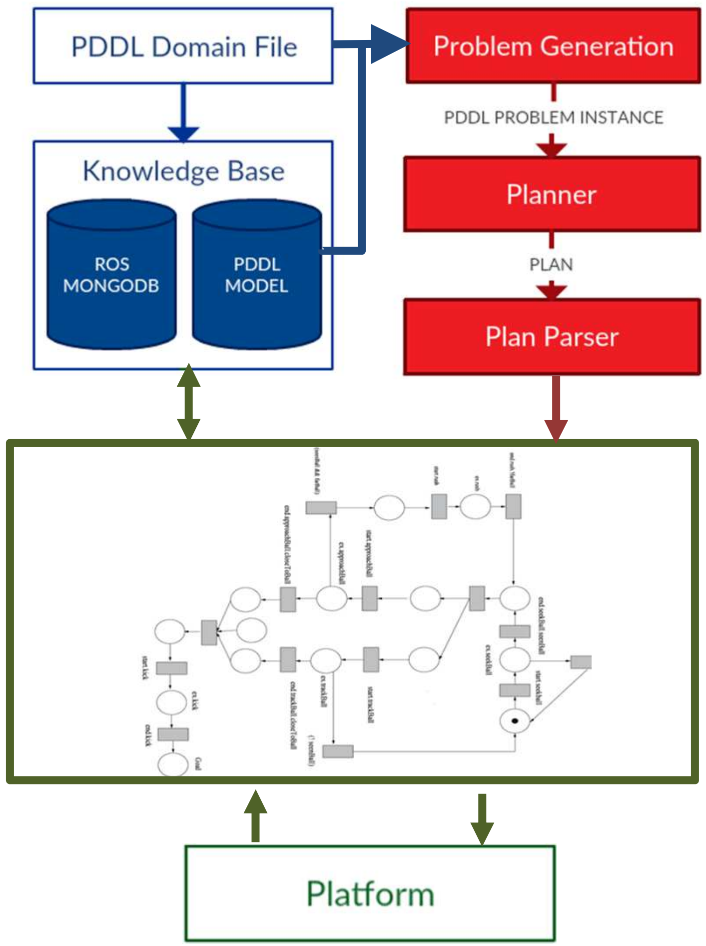
PNP execution framework



ROSPlan + PNPgen + PNP-ROS

- A proper integration of
  - o Plan generation
  - o Plan execution
  - o ROS action execution and condition monitoring

provides an effective framework for **robot planning and execution.**



Outline

- Petri Nets
- Petri Net Plans
- Execution rules
- PNP-ROS
- Demo

Petri Net definition

Definition

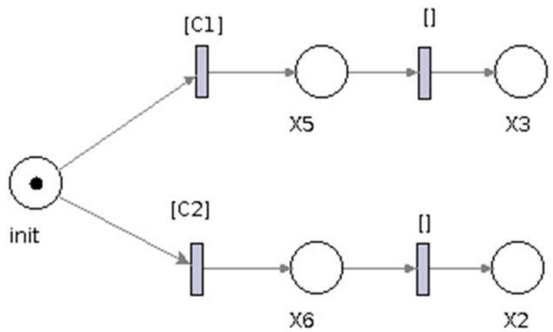
$PN = \langle P, T, F, W, M_0 \rangle$

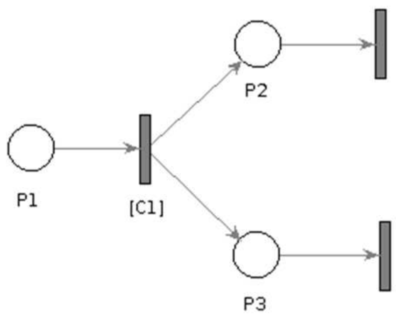
- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*.
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions*.
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of edges.
- $W : F \rightarrow \{1, 2, 3, \ldots\}$ is a weight function and $w(n_s, n_d)$ denotes the weight of the edge from n_s to n_d.
- $M_0 : P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking.
- $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$

Petri Net firing rule

Definition

1. A transition t is *enabled*, if each input place p_i (i.e. $(p_i, t) \in F$) is marked with at least $w(p_i, t)$ tokens.
2. An enabled transition may or may not fire, depending on whether related event occurs or not.
3. If an enabled transition t fires, $w(p_i, t)$ tokens are removed for each input place p_i and $w(t, p_o)$ are added to each output place p_o such that $(t, p_o) \in F$.





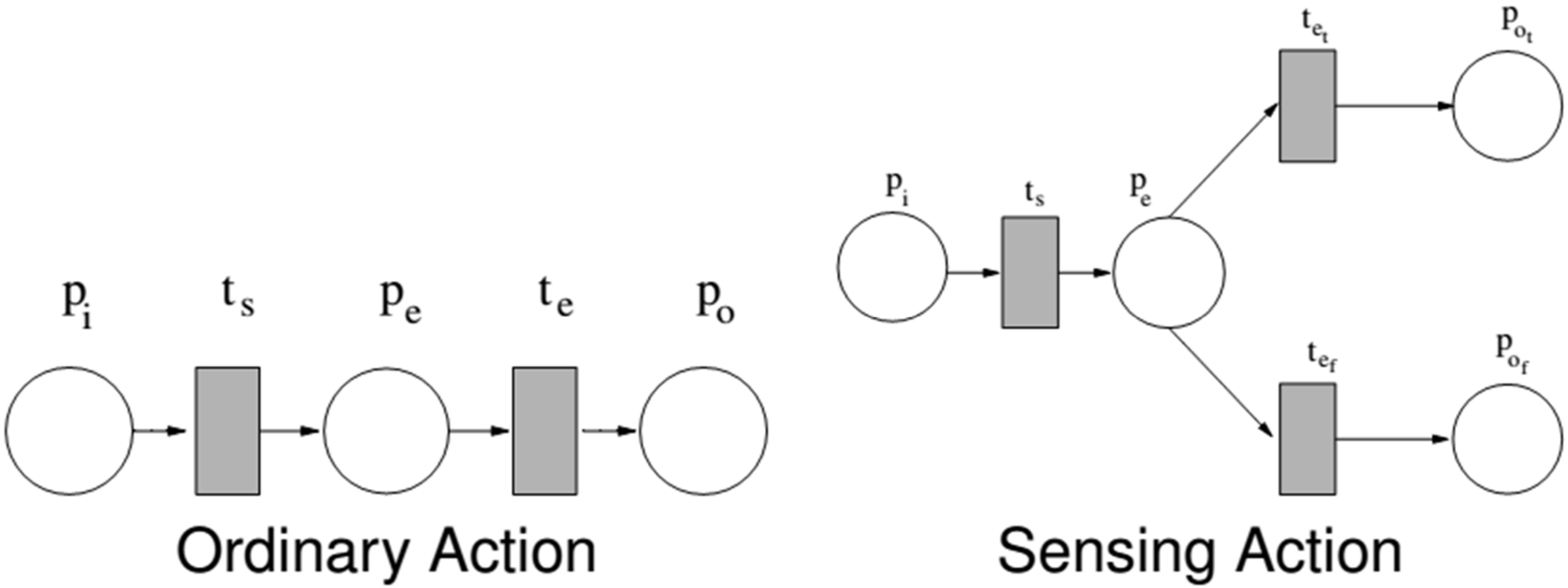
Petri Net Plans

- Petri Net Plans (PNP) are defined in terms of

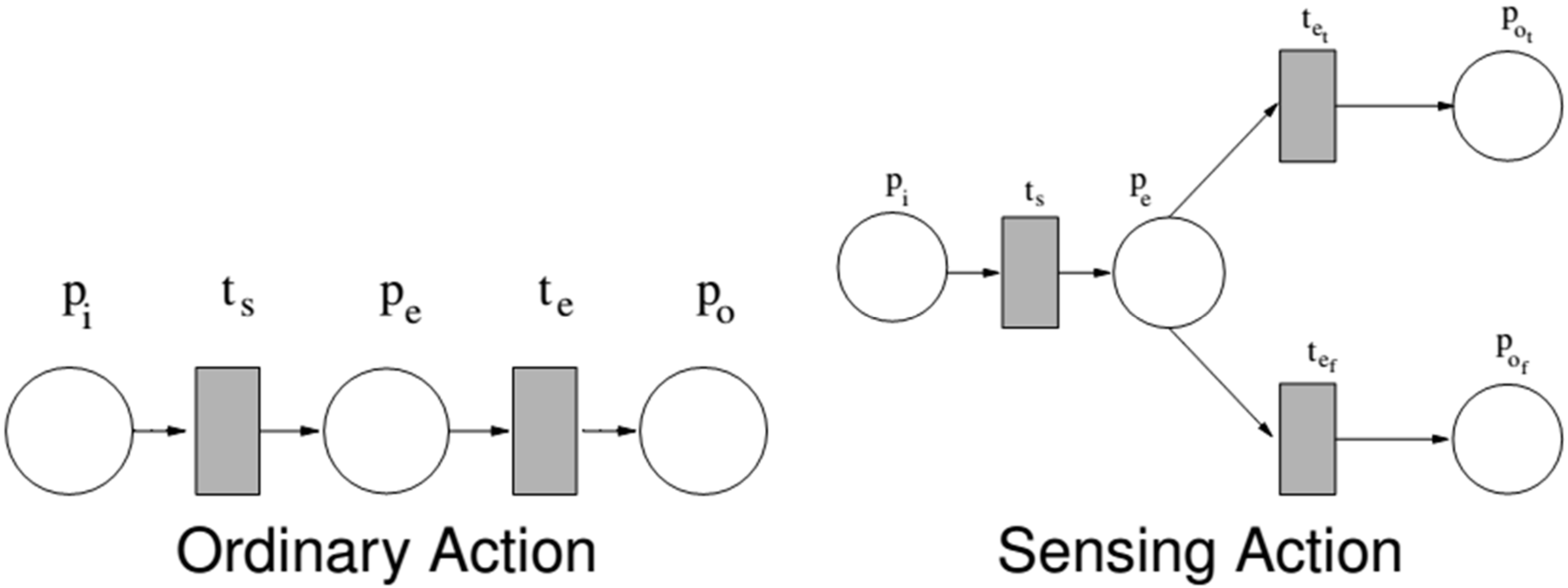
- Actions
  - ordinary actions
  - sensing actions

- Operators
  - sequence, conditional and loops
  - interrupt
  - fork/join

PNP Actions

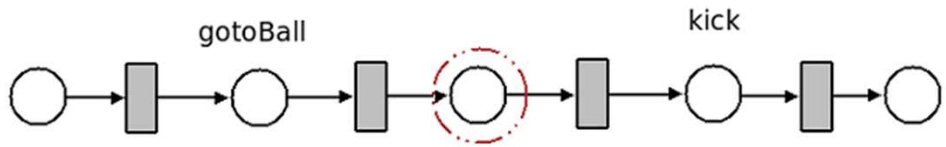

p_i
t_s
p_e
t_e
p_o
Ordinary Action
p_i
t_s
p_e
t_{e_t}
p_{o_t}
t_{e_f}
p_{o_f}
Sensing Action

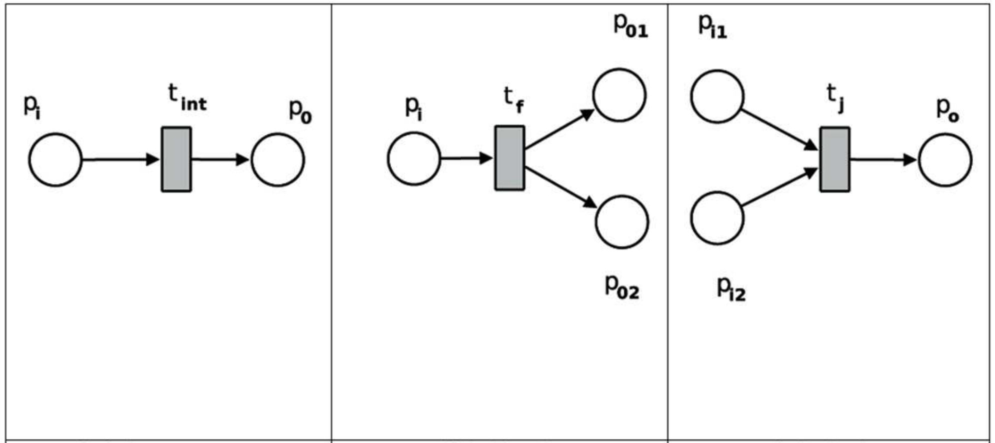
PNP Actions



p_i
t_s
p_e
t_e
p_o
Ordinary Action
p_i
t_s
p_e
t_{e_t}
p_{o_t}
t_{e_f}
p_{o_f}
Sensing Action

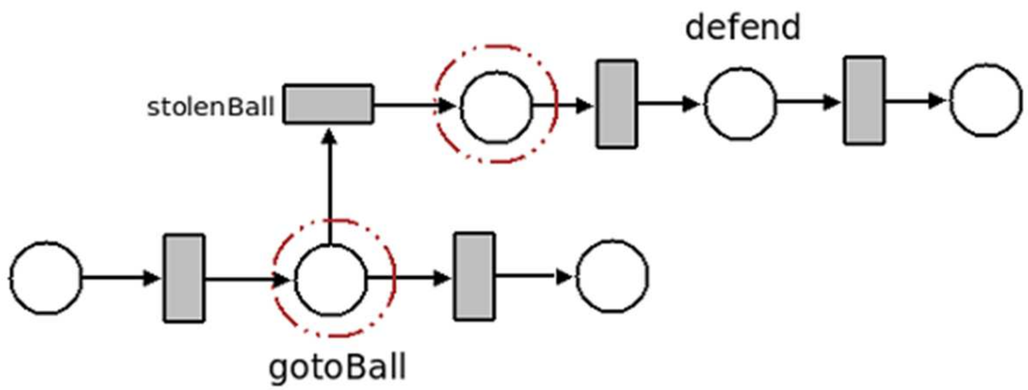
PNP Operators



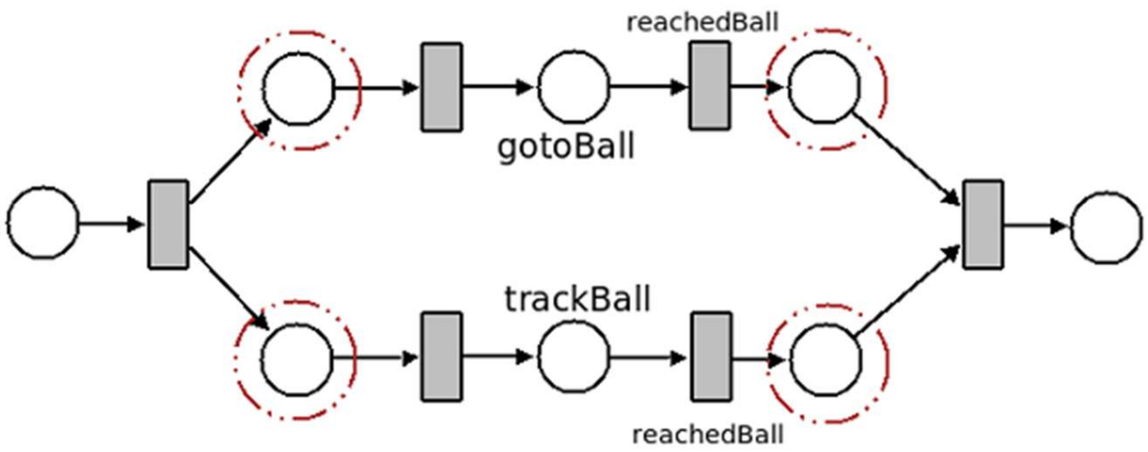


(a) Interrupt (b) Fork (c) Join

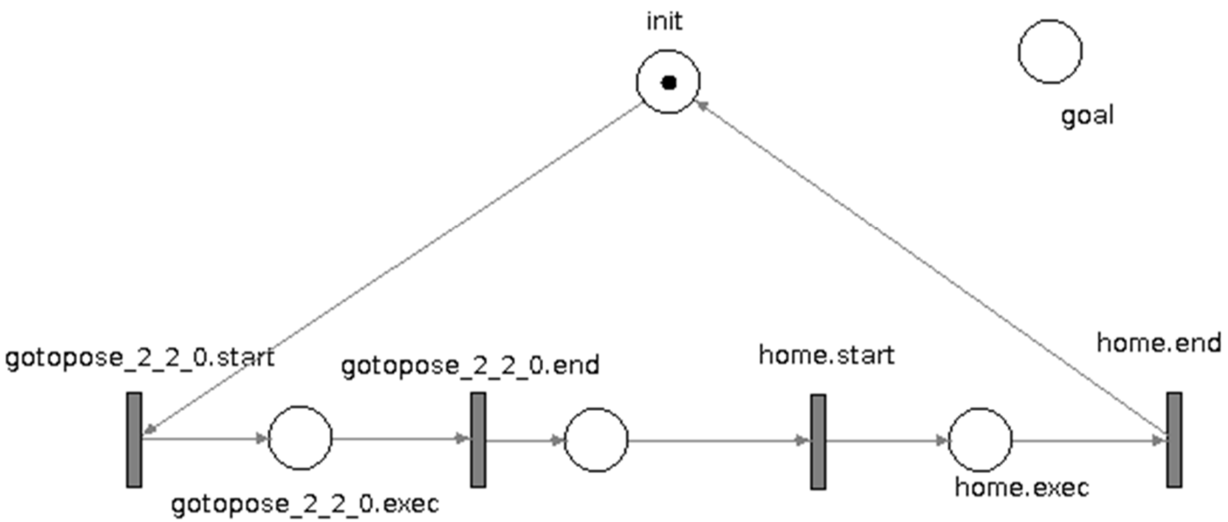
PNP interrupt



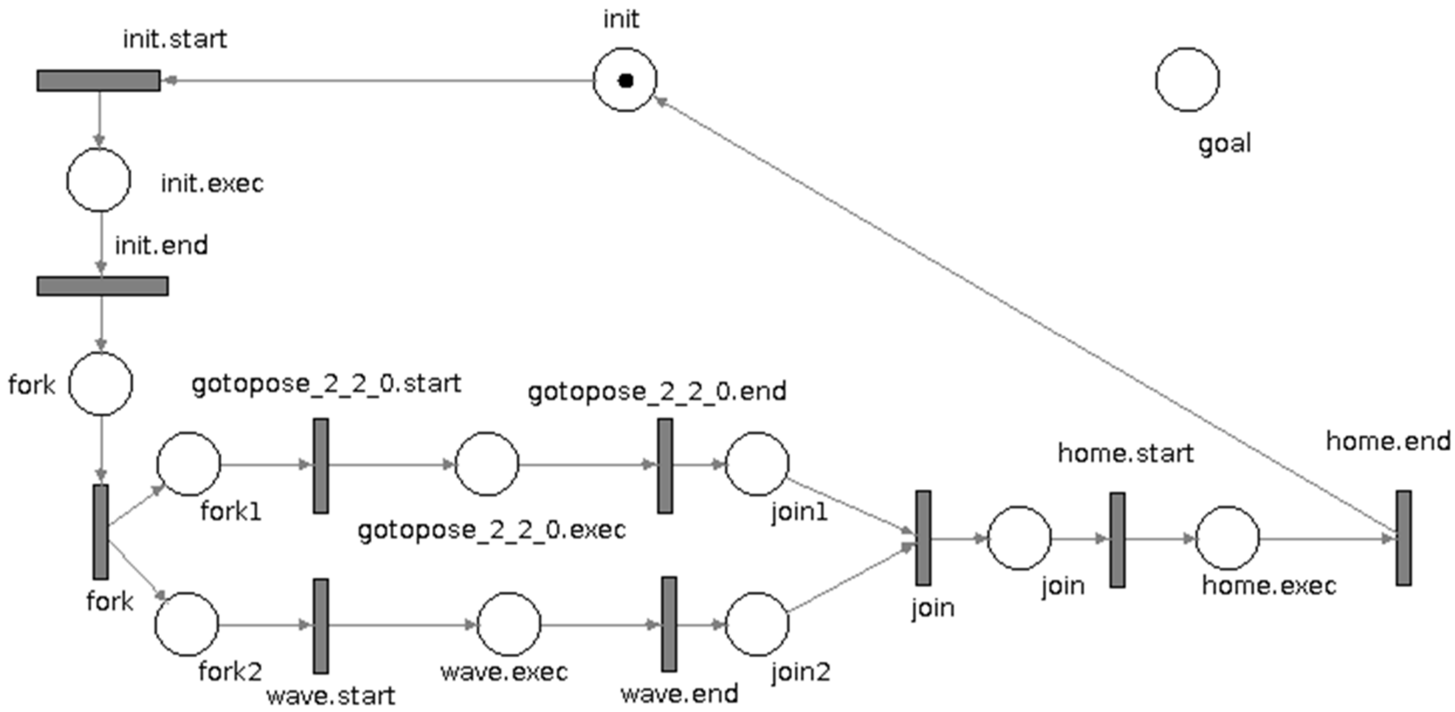
PNP concurrency



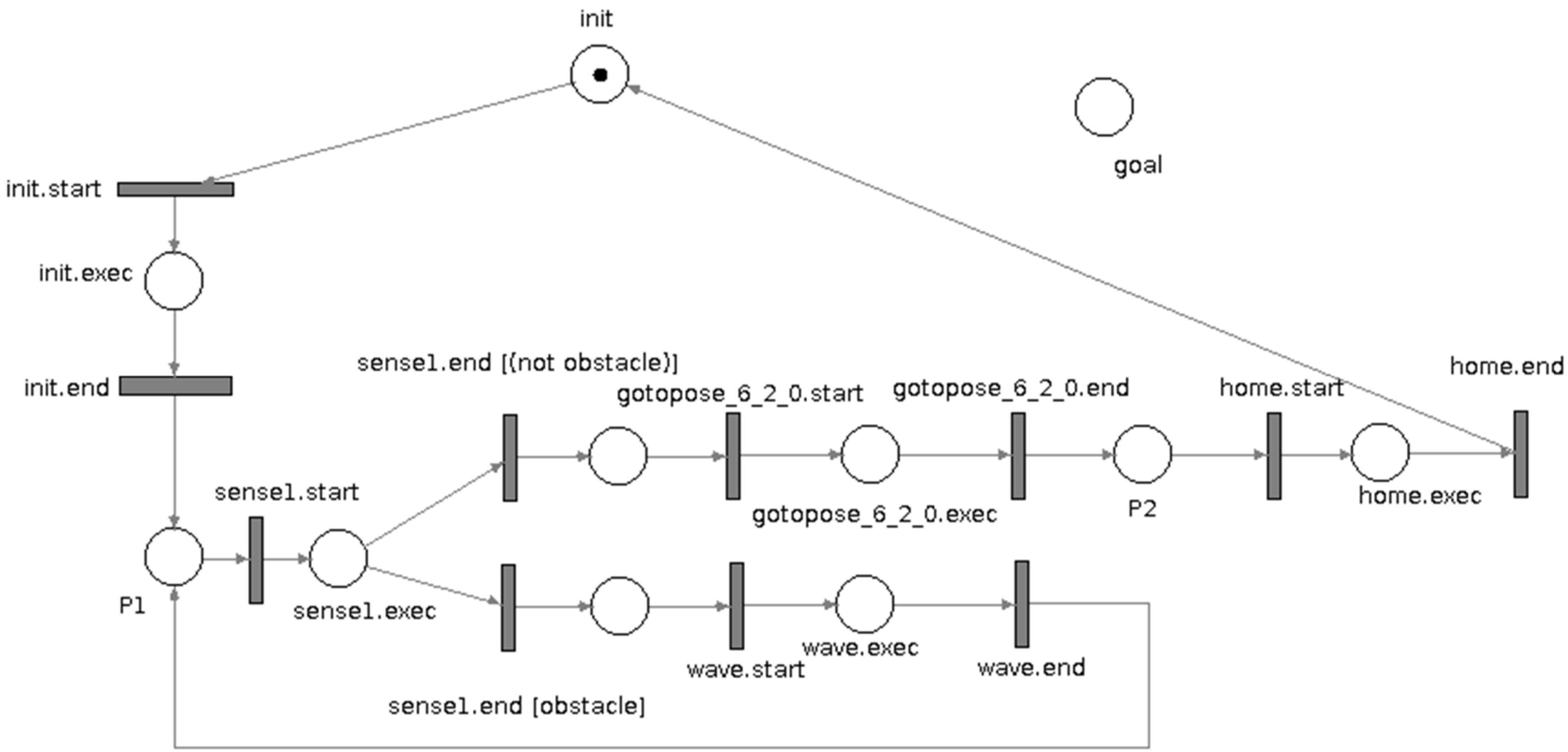
Plan 1: sequence and loop



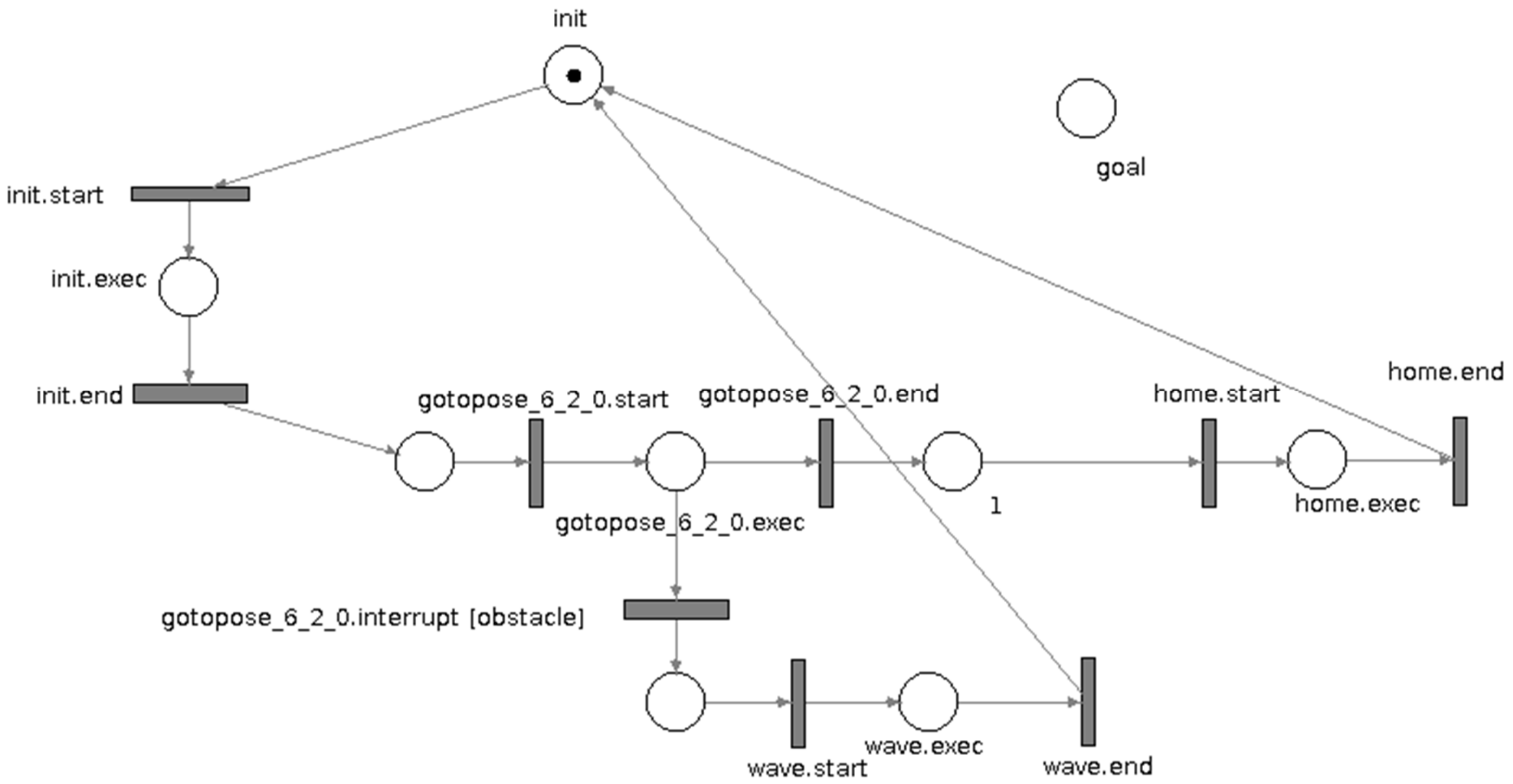
Plan 2: fork and join



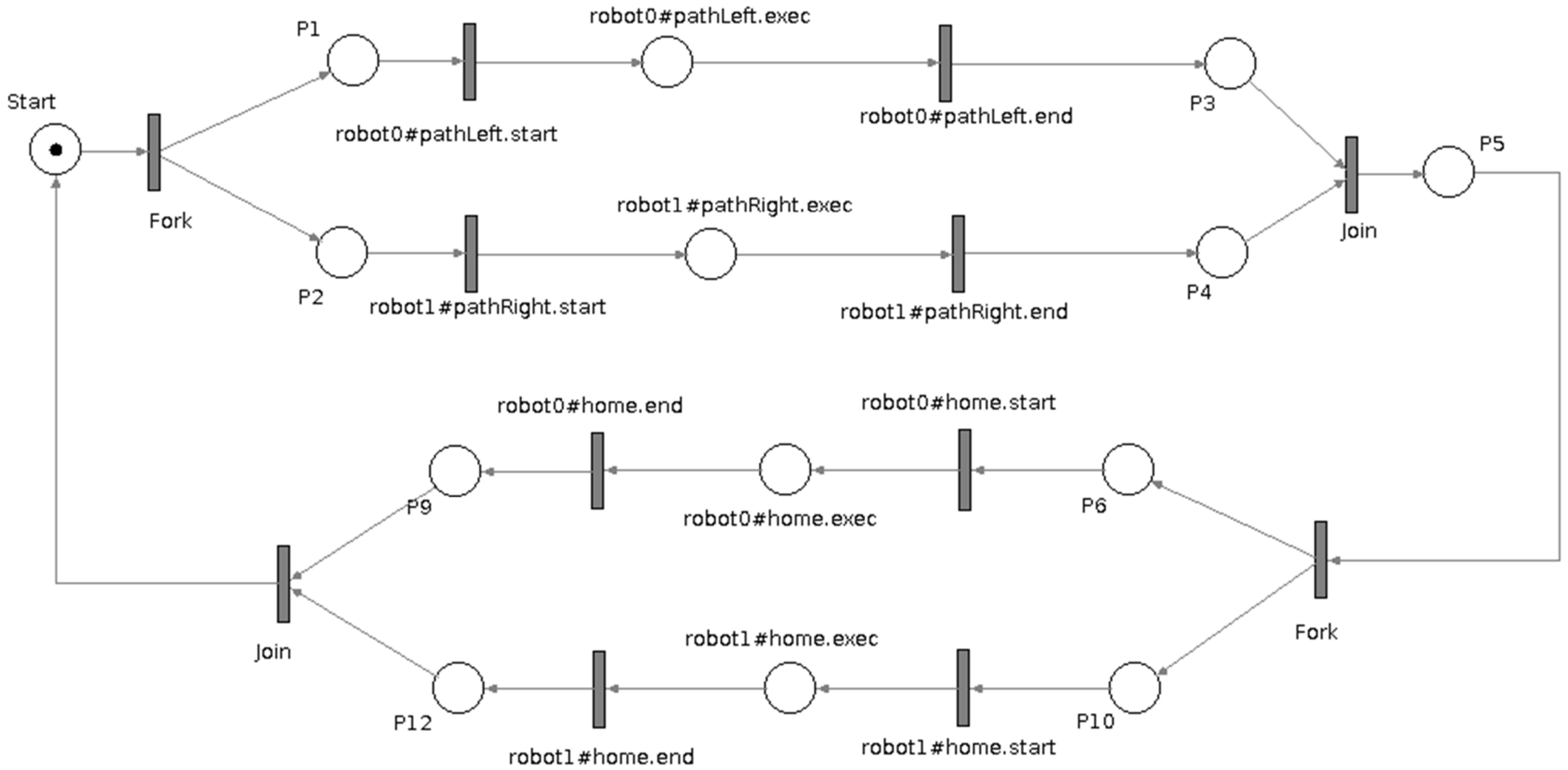
Plan 3: sensing and loop



Plan 4: interrupt



Plan 5: multi robot



PNP Execution Algorithm

procedure *execute*($PNP\ \langle P, T, F, W, M_0, G \rangle$)

$CurrentMarking = M_0$
while $CurrentMarking \notin G$ **do**
  for all $t \in T$ **do**
    if $enabled(t) \wedge KB \models t.\phi$ **then**
      $handleTransition(t)$
      $CurrentMarking = fire(t)$
    end if
  end for
end while

procedure *handleTransition*(t)

if $t.t = start$ **then**
  $t.a.start()$
else if $t.t = end$ **then**
  $t.a.end()$
else if $t.t = interrupt$ **then**
  $t.a.interrupt()$
end if

Correctness of PNP execution

- PNP execution is correct with respect to an operational semantics based on Petri nets and the robot's local knowledge.

Theorem

[ZI06] If a PNP can be correctly executed, then the Execution Algorithm computes a sequence of transitions $\{M_0, ..., M_n\}$*, such that* M_0 *is the initial marking,* M_n *is a goal marking, and* $M_i \Rightarrow M_{i+1}$*, for each* $i = 0, \ldots, n-1$.

PNP sub-plans

- Plans can be organized in a hierarchy, allowing for modularity and reuse

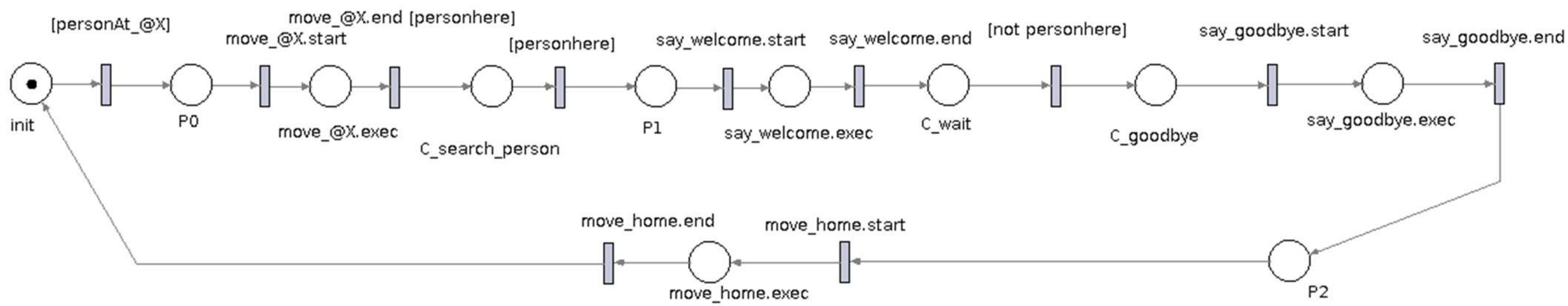
- Sub-plans are like actions:
  - when started, the initial marking is set
  - when goal marking is reached, the sub-plan ends

Plans with variables

[condition_@X] sets the value of variable X

action_@X uses the value of variable X

Example: given a condition personAt_@X, the occurrence of personAt_B115 sets the variable @X to "B115", next action goto_@X will be interpreted as goto_B115



Execution rules

Adding to the conditional plan

- interrupt (special conditions that determine interruption of an action)
- recovery paths (how to recovery from an interrupt)
- social norms
- parallel execution

Main feature

- Execution variables are generally different from the ones in the planning domain (thus not affecting complexity of planning)

Execution rules

Examples

if personhere and closetotarget **during** goto **do** skip_action

if personhere and not closetotarget **during** goto **do** say_hello; waitfor_not_personhere; restart_action

if lowbattery **during** * **do** recharge; fail_plan

after receivedhelp **do** say_thanks

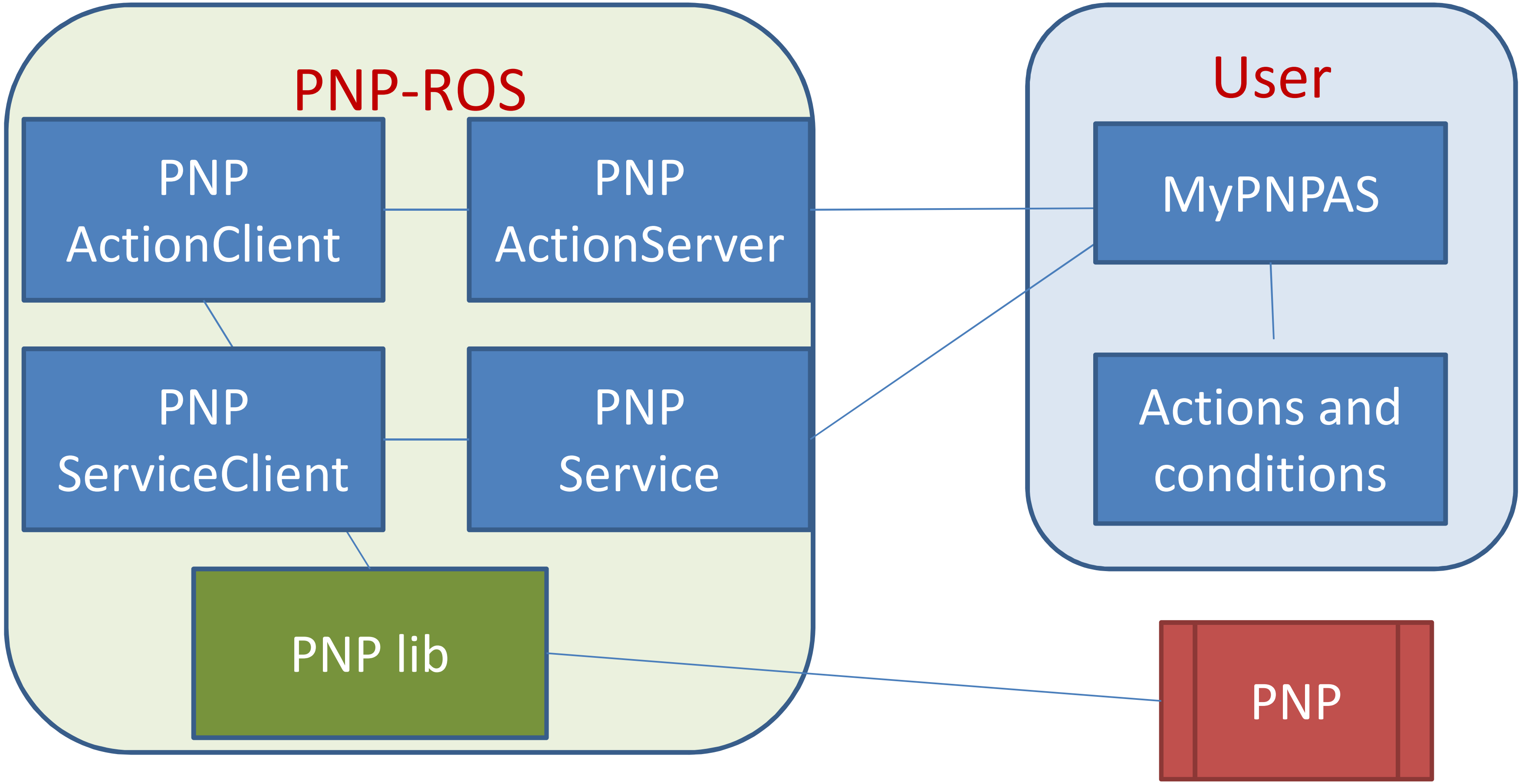
after endinteraction **do** say_goodbye

when say **do** display

PNP-ROS

- Bridge between PNP and ROS
- Allows execution of PNP under ROS using the actionlib module
- Defines a generic PNPAction and an ActionClient for PNPActions
- Defines a client service PNPConditionEval to evaluate conditions

PNP-ROS


PNP-ROS
PNP
ActionClient
PNP
ActionServer
PNP
ServiceClient
PNP
Service
PNP lib
User
MyPNPAS
Actions and
conditions
PNP

PNP-ROS

User development:

1. implement actions and conditions
2. write a PNPActionServer

PNPActionServer

```
class PNPActionServer
{
public:
    PNPActionServer();
    ~PNPActionServer();
    void start();
    // To be provided by actual implementation
    virtual void actionExecutionThread(string action_name,
                                       string action_params, bool *run);
    virtual int evalCondition(string condition); // 1: true, 0: false; -
1:unknown
}
```

PNPActionServer

```
class PNPActionServer
{
public:
    …
    // For registering action functions (MR=multi-robot version )
    void register_action(string actionname, action_fn_t actionfn);
    void register_MRaction(string actionname, MRaction_fn_t actionfn);
    …
}
```

MyPNPActionServer

```
#Include "MyActions.h"

class MyPNPActionServer : public PNPActionServer
{
   MyPNPActionServer() : PNPActionServer() {
        register_action("init",&init);
        ....


   }
}
```

MyPNPActionServer

```
PNP_cond_pub = // asynchronous conditions
      handle.advertise<std_msgs::String>("PNPConditionEvent", 10);

Function SensorProcessing
{
   …

   std_msgs::String out;
   out.data = condition;  // symbol of the condition
   PNP_cond_pub.publish(out);
}
```

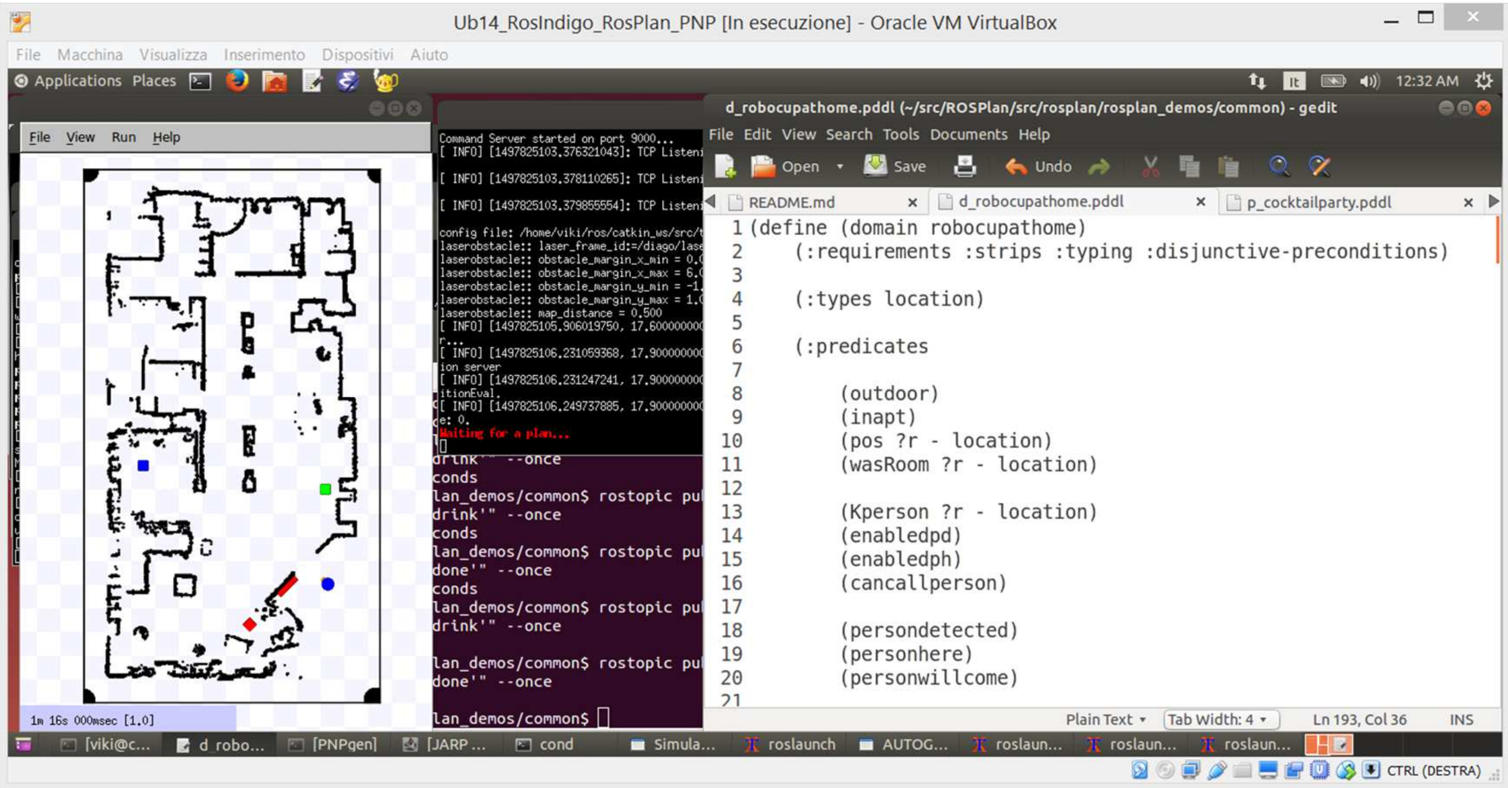
MyPNPActionServer

```
Function SensorProcessing
{
    …
   string param = "PNPconditionsBuffer/<CONDITION>";
   node_handle.setParam(param, <VALUE {1|0}>);

}
```

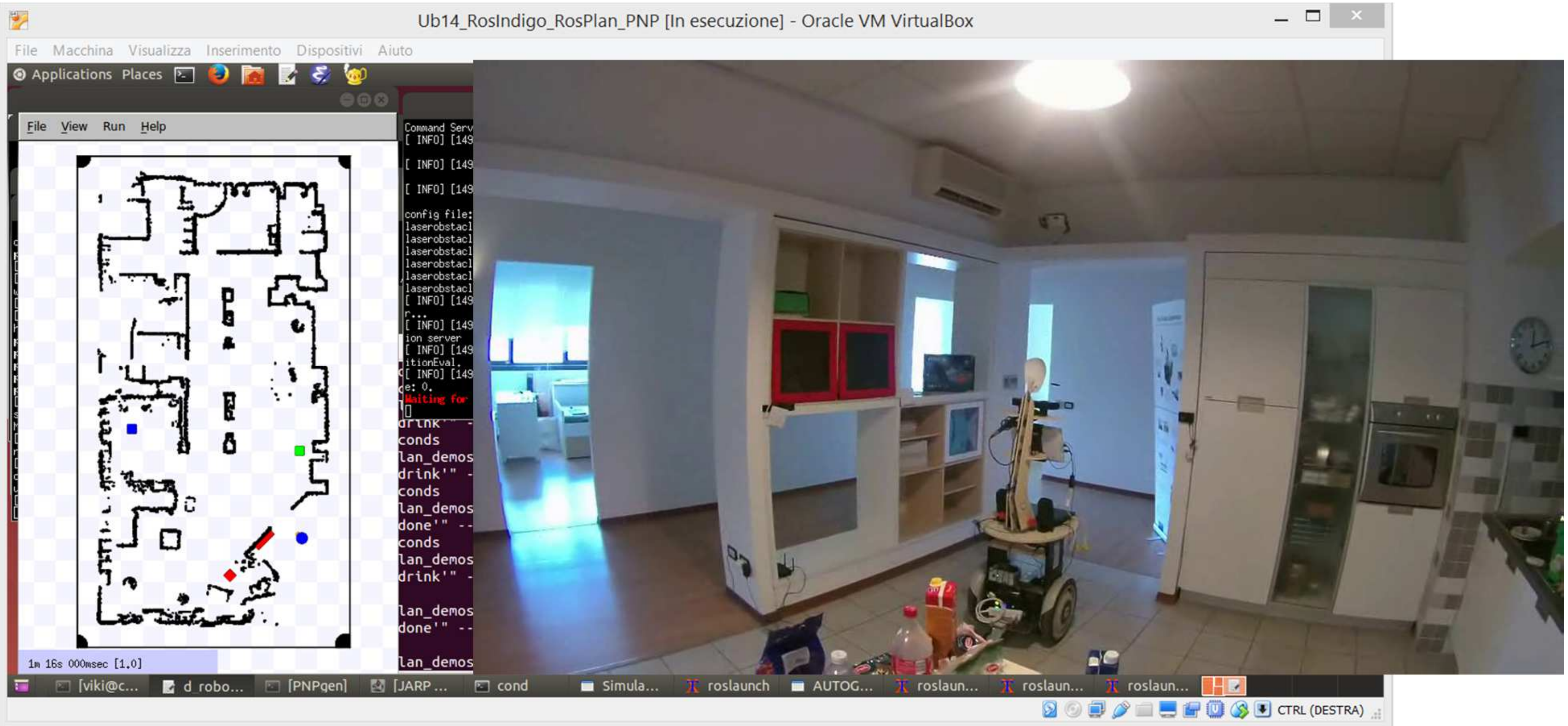
Demo

Virtual machine available in the Tutorial web site



Demo

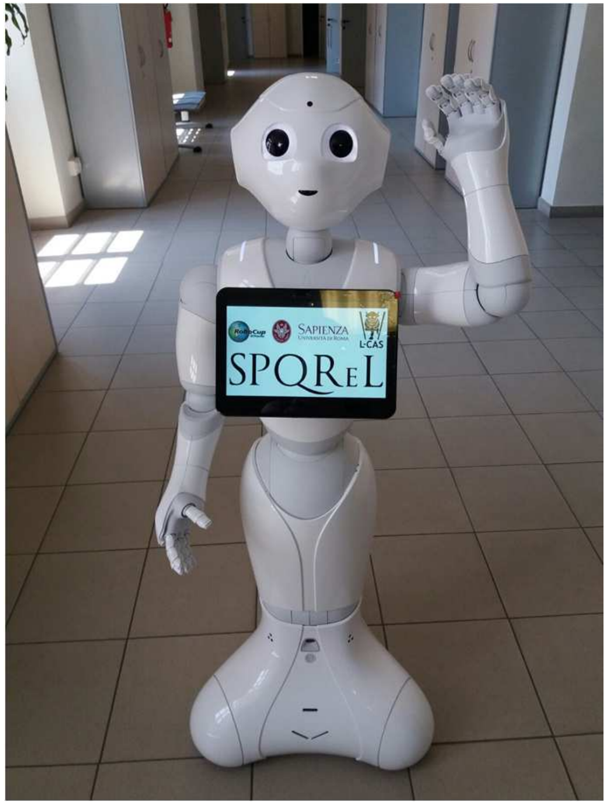
Virtual machine available in the Tutorial web site



Demo

Inspired by RoboCup@Home tasks

- RoboCup@Home domain
- Planning problems for @Home tasks
  - Navigation (rulebook 2016)
  - Cocktail Party (rulebook 2017)



NOTE: We are using this framework in our SPQReL team that will compete in RoboCup@Home 2017 SSPL

References

- **Petri Net Plans - A framework for collaboration and coordination in multi-robot systems**. V. A. Ziparo, L. Iocchi, Pedro Lima, D. Nardi, P. Palamara. Autonomous Agents and Multi-Agent Systems, vol. 23, no. 3, 2011.
- **Dealing with On-line Human-Robot Negotiations in Hierarchical Agent-based Task Planner**. E. Sebastiani, R. Lallement, R. Alami, L. Iocchi. In Proc. of International Conference on Automated Planning and Scheduling (ICAPS), 2017.
- **Short-Term Human Robot Interaction through Conditional Planning and Execution.** V. Sanelli, M. Cashmore, D. Magazzeni, L. Iocchi. In Proc. of International Conference on Automated Planning and Scheduling (ICAPS), 2017.
- **A practical framework for robust decision-theoretic planning and execution for service robots.** L. Iocchi, L. Jeanpierre, M. T. Lazaro, A.-I. Mouaddib. In Proc. of International Conference on Automated Planning and Scheduling (ICAPS), 2016.
- **Explicit Representation of Social Norms for Social Robots.** F. M. Carlucci, L. Nardi, L. Iocchi, D. Nardi. In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), 2015.