PNP-ROS

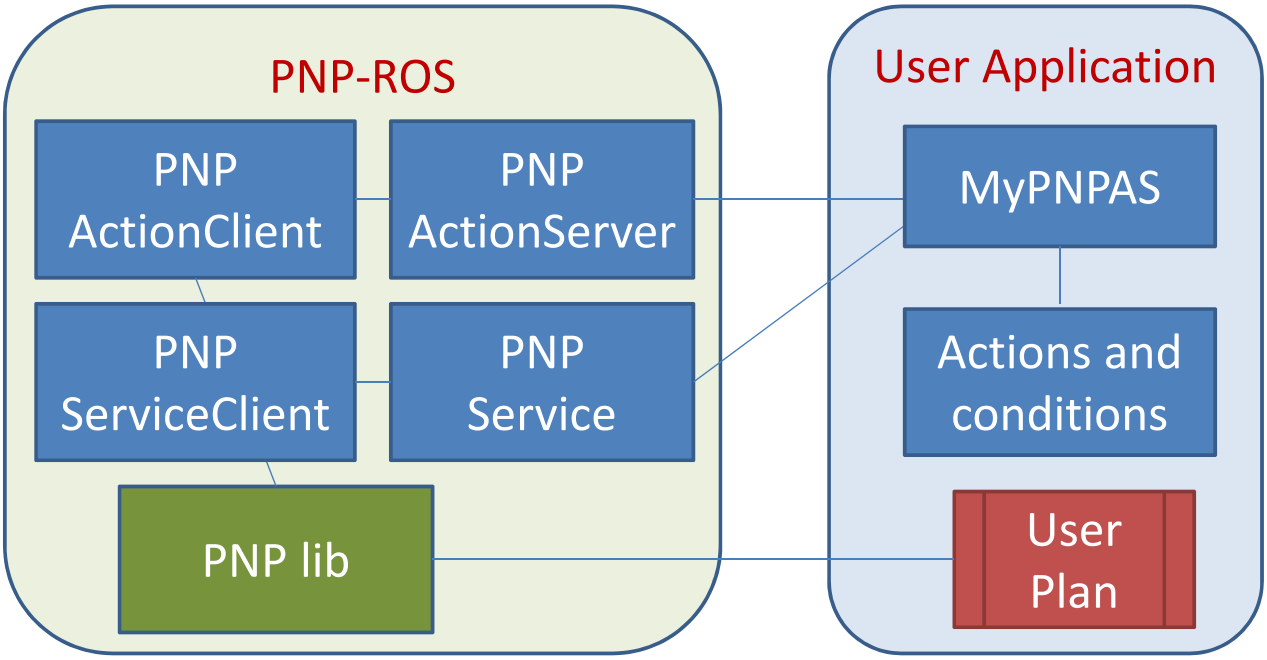
SAPIENZA
UNIVERSITÀ DI ROMA

Luca Iocchi
Dipartimento di Ingegneria Informatica
Automatica e Gestionale

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

User development:

1. implement actions and conditions – Writing a PNPActionServer (not required for lab AI course)
2. write plans – Using Jarp

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

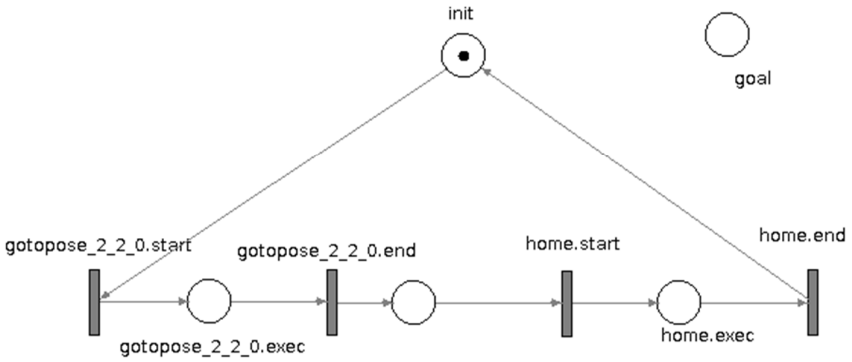
```
Function SensorProcessing
{
    …
    string param = “PNPconditionsBuffer/<CONDITION>”;
    node_handle.setParam(param, <VALUE {1|0}>);

}
```


Writing plans with Jarp

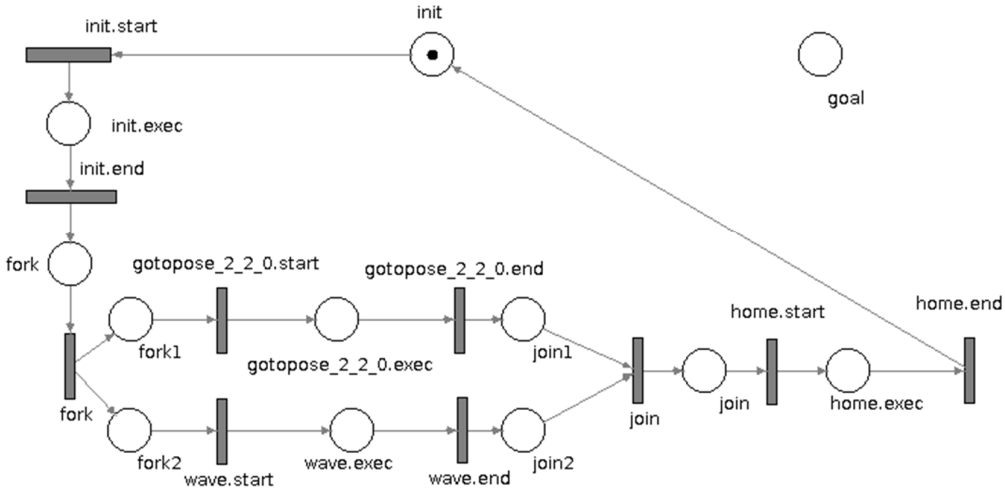
Jarp is a graphical interface to write Petri Nets, that can be used to write PNPs

Available in the Jarp directory of PetriNetPlans repository

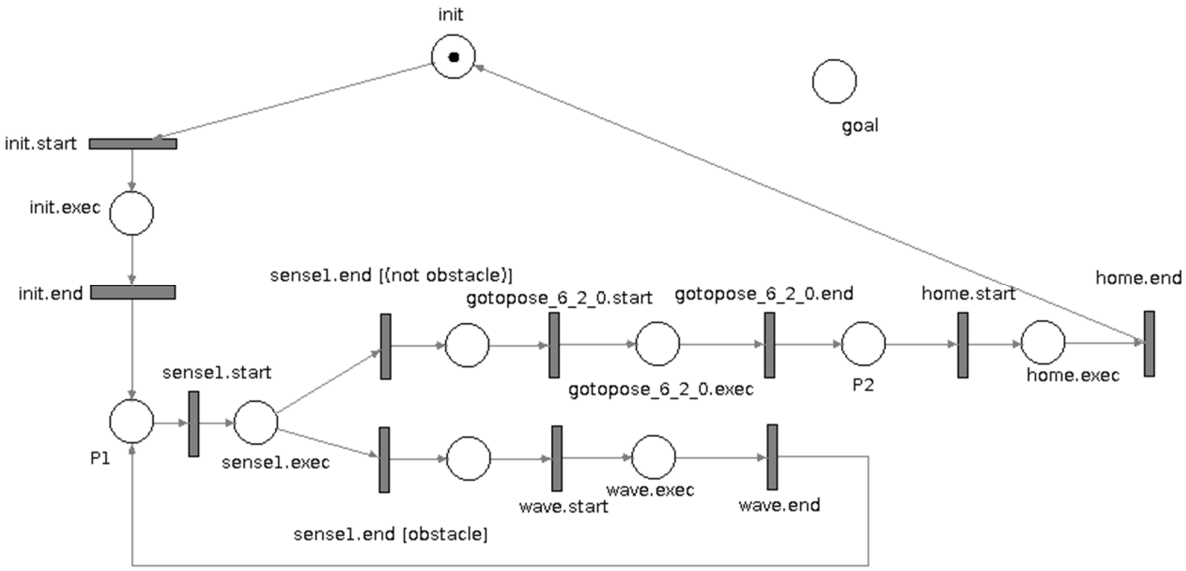
Plan 1: sequence and loop



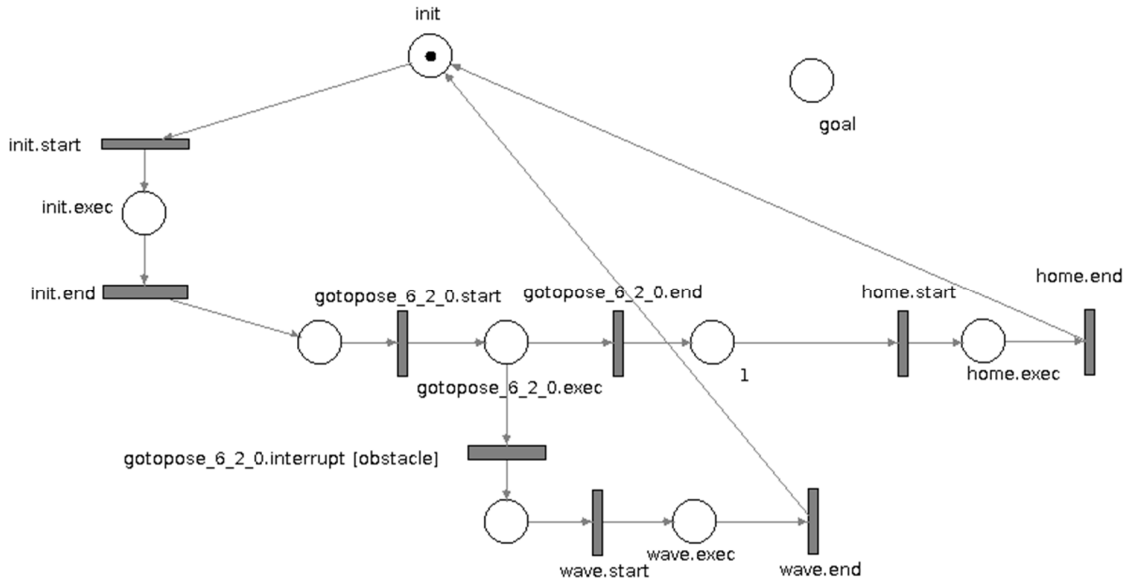
Plan 2: fork and join



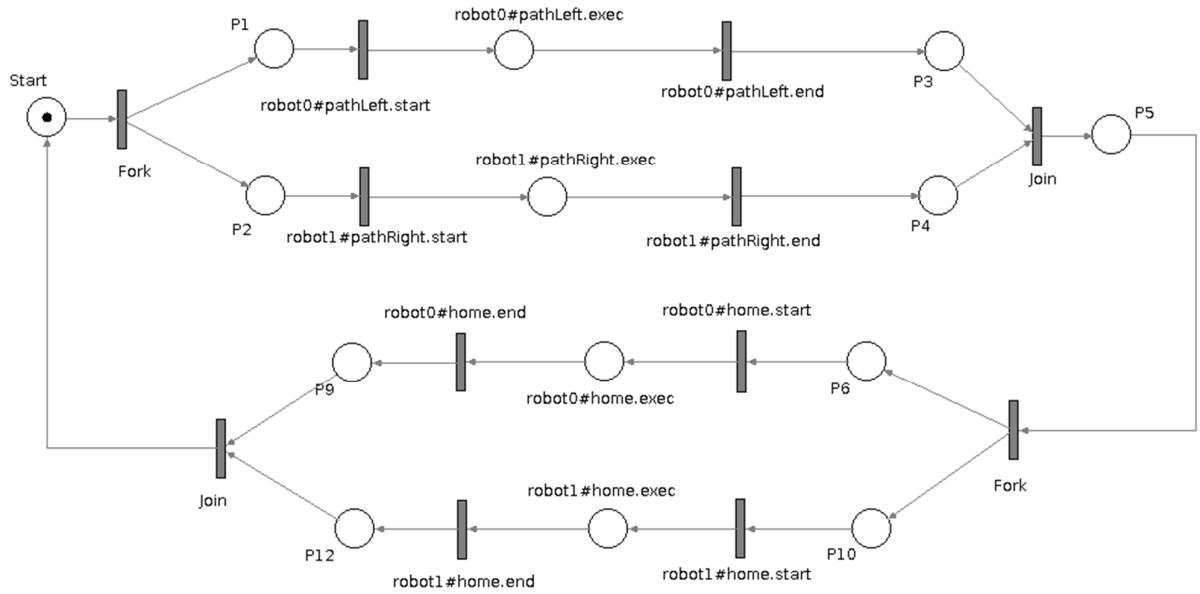
Plan 3: sensing



Plan 4: interrupt



Plan 5: multi robot



Write plans as text files

PNPs can also be specified as text files.

https://github.com/iocchi/PetriNetPlans/wiki/Plan-syntax-for-PNP-generation

Examples

```
A; B; C;
< phi? A; B; C : (not phi)? D; E >
```

Generating PNP from plan text files

PNPgen library contains tools to transform plans specified as text files (in different formats) into PNPs

pnpgen_translator inline <planname.plan> [executionrules]

Generating PNP from automated planners

PNP is integrated with ROSPlan for using a PDDL planner to generate plans and transform them into PNPs for execution

http://kcl-planning.github.io/ROSPlan/

ICAPS 2017 Tutorial on AI Planning for Robotics and Human-Robot Interaction

http://kcl-planning.github.io/ROSPlan/demos/conference_pages/tutorialICAPS2017.html

Execution rules

Execution rules are rules evauated at run time that affects the execution of a plan.

if condition *during* action *do* recoveryplan

Execution rules

PNPgen integrates execution rules when generating a PNP.

goto_A;
goto_B; ! *if* obstacle *do* goto_Home; fail_plan !
goto_C;

Example

Write a sequential plan with execution rules to implement this behavior

- search a known area (visit a sequence of pre-defined target goals)
- If a person is encountered during the path, say hello and continue the plan
- If a large obstacle is encountered, go to the office room and report the issue

Available conditions: personhere, obstaclehere

Available actions: goto_<target>, say_<something>