# Modelling, Analysis and Execution of Robotic Tasks using Petri Nets

Hugo Costelha and Pedro Lima

*Abstract*— **This paper introduces Petri net based models of robotic tasks, which can be used to analyse and synthesise task plans, taking into account a Petri net model that abstracts the relevant features from the robot environment as well. Logical analysis concerning deadlocks and resource conservation can be performed over the ordinary version of the model. A task plan modeled by a Petri net can be extracted from the generalised stochastic version of the model, representing the optimal plan given a probabilistic measure of uncertainty associated to the effects of its composing actions. The Petri net representing the model is suitable for being ran directly within the code, as well as for plan monitoring during execution time. Simulation results illustrating the methodology are presented for a robotic soccer scenario.**

## I. INTRODUCTION

Most of the existing robotic task models are not based on formal approaches but tailored to the task at hand, usually leading to task plans with few actions. Applying discrete event system concepts to model robotic tasks provides a systematic approach to modelling, analysis and design, scaling up to realistic applications, and enabling analysis of formal properties, as well as design from specifications.

The need for a modular design of complex robotic tasks from specifications of quantitative and qualitative properties is the main motivation for this work. Petri nets come up as an adequate modelling and analysis tool to accomplish this goal. Their modelling and analysis power applied to single or multi robot tasks definition can prove to be quite effective.

Most of the work found on the literature concerning the design of robotic tasks using Discrete Event Systems (DES) is based on FSA for code generation [3], qualitative specifications [4], some quantitative specifications [5], modularisation [4] and even to model multi-robot systems [6], [8]. Work using Petri nets to design robotic tasks under temporal requirements, focusing also on the generation of real-time, error-free code can be found in [9]. Recently Petri net Plans [10] were introduced for design and execution of task plans.

The work described here can be seen as an extension of the ideas introduced in [6] and [11]. The latter proposes a framework for qualitative and quantitative performance analysis using Petri nets.

Petri nets are preferred to Finite State Automata (FSA) due to their larger modelling power [7] and because one can model the same state space with a smaller graph. Moreover, although composition of Petri nets usually leads to an exponential growth in the state space (as for FSA), graphically the growth is linear in the size of the composed graphs. This makes the design process simpler for the task designer, and helps managing the display of the tasks both for monitoring and designing purposes.

Another reason to choose Petri nets is its modularity, as each resource can be easily modeled separately and then composed with others. Although composition operators exist for FSA, Petri nets can model subsystems with input and output places, so that they can be connected as in a circuit. Additionally, Petri nets are suited to model concurrency, parallelism, synchronisation and decision making.

The main difference between our work and past work is that we do not just create the high-level models for execution purposes, but rather model the action itself using generalised stochastic Petri nets, leading to a more realistic model which also enables obtaining not only qualitative properties of the task, but also performance measures. Thus, plans can be analysed and synthesised automatically as explained later in the paper. Throughout this paper we will use examples from a robotic soccer scenario.

## II. PETRI NETS

### *A. MARKED ORDINARY PETRI NETS*

The simplest models we use are Marked Ordinary Petri nets [7]:

*Definition 2.1:* A marked ordinary Petri net (MOPN) is a five-tuple $PN = \langle P, T, I, O, \mathcal{M}_0 \rangle$, where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite, not empty, set of places;
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions;
- $I = P \times T$ represents the arc connections from places to transitions, such that $i_{lj} = 1$ if, and only if, there is an arc from $p_l$ to $t_j$, and $i_{lj} = 0$ otherwise;
- $O = T \times P$ represent the arc connections from transition to places, such that $o_{lj} = 1$ if, and only if, there is an arc from $t_l$ to $p_j$, and $o_{lj} = 0$ otherwise;
- $\mathcal{M}_j = \{m_{1_j}, \ldots, m_{n_j}\}$ is the state of the net, and represents the marking of the net at time $j$, where $m_{n_j} = q$ means there are $q$ tokens in place $p_n$ at time instant $j$. $\mathcal{M}_0$ is the initial marking of the net.

In this class of Petri nets, all the transitions are *immediate* (have zero firing time), i.e., once they are enabled and fired, the new marking is instantly reached.

### *B. GENERALISED STOCHASTIC PETRI NETS*

MOPNs are suited for qualitative analysis, but not for performance analysis. For this purpose, one can use generalised stochastic Petri nets (GSPN) [1].

*Definition 2.2:* A GSPN is an eight-tuple $PN = \langle P, T, I, O, \mathcal{M}_0, R, S \rangle$, where:

H. Costelha and P. Lima are with Instituto Superior Técnico, Universidade Técnica de Lisboa 1049-001 Lisboa, Portugal {hcostelha,pal}@isr.ist.utl.pt

- $P, T, I, O, \mathcal{M}_0$ are as defined in 2.1;
- $T$ is partitioned in two sets: $T_I$ of immediate transitions and $T_E$ of exponential transitions;
- $R$ is a function from the set of transitions $T_E$ to the set of real numbers, $R\left(t_{E_j}\right) = \mu_j$, where $\mu_j$ is called the firing rate of $t_{E_j}$;
- $S$ is a set of *random switches*, which associate probability distributions to subsets of conflicting immediate transitions.

Stochastic (exponential) transitions, once enabled, fire only when an exponentially disributed time $d_j$ has elapsed. This definition of GSPNs includes also the possibility of associating a probability distribution to conflicting immediate transitions, by the use of the *random switches*.

The GSPN marking is a semi-Markov process with a discrete state space given by the reachability graph of the net for an initial marking [1]. A Markov chain can be obtained from the marking process, and the transition probability matrix computed by using the firing rates of the exponential timed transitions and the probabilities associated with the random switches.

Given that the marking of the GSPN is equivalent to a Markov chain allows the use of tools already available to analyse Markov chains directly with the GSPN, instead of relying on e.g., Monte Carlo simulation. Thus, we have two different (complementary) types of analysis that can be performed with GSPN: conservation properties (based on T-invariants and P-invariants) and performance evaluation (based on the continuous time, discrete state space Markov process).

## III. MODELLING ROBOTIC TASKS USING PETRI NETS

Our robotic task models are divided in layers with different degrees of abstraction. The layers are (from the lowest to the highest level of abstraction):

1) **Environment**: Includes environment related models;
2) **Action Executor**: Here one finds the action models;
3) **Action Coordinator**: Includes the task plan models, used in the decision process;
4) **Organisation**: The higher-level layer, where roles are dynamically assigned and mission goals selected.

Layers 2, 3 and 4 are tightly coupled with what runs on the robot, while the Environment layer is only used for analysis purposes. In this paper we are not concerned with the Organisation layer, but focus on the other three layers. Each layer includes several models (e.g., in the Action Executor we have one model per action), and each model can be designed separately, thus simplifying the design process.

### A. THE ENVIRONMENT LAYER

The Environment layer includes the environment state model, including state changes that might occur due to actions performed either by the robot, by other robots, or other agents. A discrete set of relevant states is abstracted from the actual environment.

The core of the environment model is the *predicate* (e.g.: SEE_BALL), consisting of a logic with an output value of $true$ or $false$. In order to define a Petri net model of a predicate, we need first to define a *predicate place*:

*Definition 3.1:* A predicate place $p$ is a place associated with the predicate $P()$, described by $p \models P()$, such that:

- $\forall_j, P_j() = true \Leftrightarrow m_{p_j} = 1$
- $\forall_j, P_j() = false \Leftrightarrow m_{p_j} = 0$,

where $P_j()$ is the predicate $P()$ at time step $j$.

*Definition 3.2:* Given a predicate place $p$ associated with the predicate $P()$, $\neg p$ is the place associated with the predicate $\neg P()$, such that:

- $\forall_j, P_j() = true \Leftrightarrow m_{\neg p_j} = 0$
- $\forall_j, P_j() = false \Leftrightarrow m_{\neg p_j} = 1$,

where $P_j()$ is the predicate $P()$ at time step $j$.

Given Definitions 3.1 and 3.2 we obtain Lemma 3.1.

*Lemma 3.1:* Given a predicate place $p$ associated with predicate $P()$, it results that

$$\forall_j, m_{p_j} + m_{\neg p_j} = 1,$$

i.e., the places are mutually exclusive.

Given the definition of a predicate place, we can build the full Petri net associated with the given predicate.

*Definition 3.3:* A Petri net model of a predicate is a MOPN, where:

- $P = \{\neg p, p\}$, where $p$ is a predicate place associated with predicate $P()$;
- $I = \emptyset$;
- $O = \emptyset$;
- $\forall_j \mathcal{M}_j = \{0, 1\} \vee \{1, 0\}$.

Having defined the predicate models, we can define the environment models:

*Definition 3.4:* A Petri net model of the environment is a GSPN, where:

- $\forall_{p_i} \exists_{P_n()}, p_i \models P_n()$, every place is a predicate place;
- $\forall_j \exists_l \exists_m : i_{lj} = 1 \wedge o_{jm} = 1$, all transitions have at least one input arc and one output arc;

Basically, *Environment* models consist of GSPNs, where all the places are associated with a given predicate, and the underlying MOPN is conservative, i.e., the total number of tokens is constant. An environment model example of the ball location in a soccer field is depicted in Fig. 1.

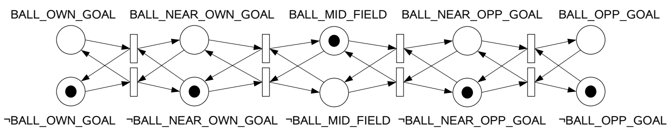

Fig. 1. Part of an *Environment* model (ball location).

### B. THE ACTION EXECUTOR LAYER

The first, and lowest level layer of the group of layers representing the robot actions, is the *Action Executor* layer, which holds the action models.

An action is mainly described by the effects it causes on the environment and the conditions that need be met for the effects to take place. The complete action model is achieved by specifying the *Running-conditions*, *Effects*, and the relation between them. The *Effects* are further divided in *Success Effects* and *Failure Effects*. The meaning of these sets is as follows:

- **Running-conditions**: Conditions that need be met for the action to be able to provoke changes in the world;
- **Effects**: Composed of *Success Effects* and *Failure Effects*, describe the action impact on the environment;
  - **Success Effects**: Effects associated with the success of the action. These include the *desired effects* plus additionally intermediate effects that might occur in order to achieve success. The action can have different subsets of *Success Effects*, but only one subset of *desired effects*;
  - **Failure Effects**: Relevant undesired effects, which may happen when the action is running. The action can have different subsets of *Failure Effects*.

*Definition 3.5:* A Petri net model of an action is a GSPN, where:

1) $P = P_A \cup P_E \cup P_R$ with $P_A \cap P_E = \emptyset$ and $P_A \cap P_R = \emptyset$, where

| | |
|---|---|
| $P_A$ | is the action place set; |
| $P_E$ | is the effects place set; |
| $P_R$ | is the running-conditions place set; |

2) $P_A = \{a\}$ consists on a single place, the *action place*, which is an input place (drawn with a double circle);
3) $(P_E \cup P_R) \subseteq Predicates$ and $P_A \nsubseteq Predicates$ , all places are predicate places, except for the *action place*;
4) $P_E = P_{E_S} \cup P_{E_F}$, where $P_{E_S}$ and $P_{E_F}$ are designated respectively *success places set* and *failure places set*.
5) $P_{E_S} = P_{E_{S_I}} \cup P_{E_{S_D}}$, where $P_{E_{S_I}}$ and $P_{E_{S_D}}$ are designated respectively *intermediate effects place set* and *desired effects place set*.
6) $T = T_S \cup T_F$ with $T_S \cap T_F = \emptyset$, where:

| | |
|---|---|
| $T_S$ | is the set of transitions associated with successful impact of the action; |
| $T_F$ | is the set of transitions associated with failure impact of the action; |

7) $\forall_{t_j \in T} \forall_{p_n \in P \setminus P_A}, i_{nj} = 1 \Rightarrow (o_{jm} = 1) \wedge [(p_m = \neg p_n) \vee (m = n)]$. If there is an arc from place $p_n$, associated to predicate $P()$, to transition $t_j$, then there is an arc from $t_j$ to place $p_m$, associated to predicate $\neg P()$, or an arc back to $p_n$;
8) $\forall_{t_j \in T} \forall_{p_n \in P_A}, i_{nj} = 1 \wedge o_{jn} = 1$. All transitions have one input arc and one output arc to the action place;
9) $\forall_{t_j \in T} \forall_{p_n \in P_R}, i_{nj} = 1$. All transitions have one input arc from each running-condition;
10) $\forall_{t_j \in T_S}$, if $\exists_{p_n \in P_{E_{S_D}}}, o_{jn} = 1$ then $\forall_{p_m \in P_{E_{S_D}}}, o_{jm} = 1$. If a desired effect place is an output place of a transition, then all the desired effects places are also output places of that transition;
11) $\forall_{t_j \in T_S}$ the transition label is *success*;
12) $\forall_{t_j \in T_F}$ the transition label is *failure*;

*Lemma 3.2:* Item 7 of Definition 3.5 implies that the action model fulfils does not violate the predicate places definition and associated lemma for all the predicate places used in the action model.

Modelling an action consists in building a GSPN following the rules stated in Definition 3.5. The general Petri net model of an action is depicted in Fig. 2 (the enabling arcs are depicted in a different colour just to increase readability). Note that both the *action place* and *Running-conditions* places, since they are connected to all the transitions, function as *enabling places*.

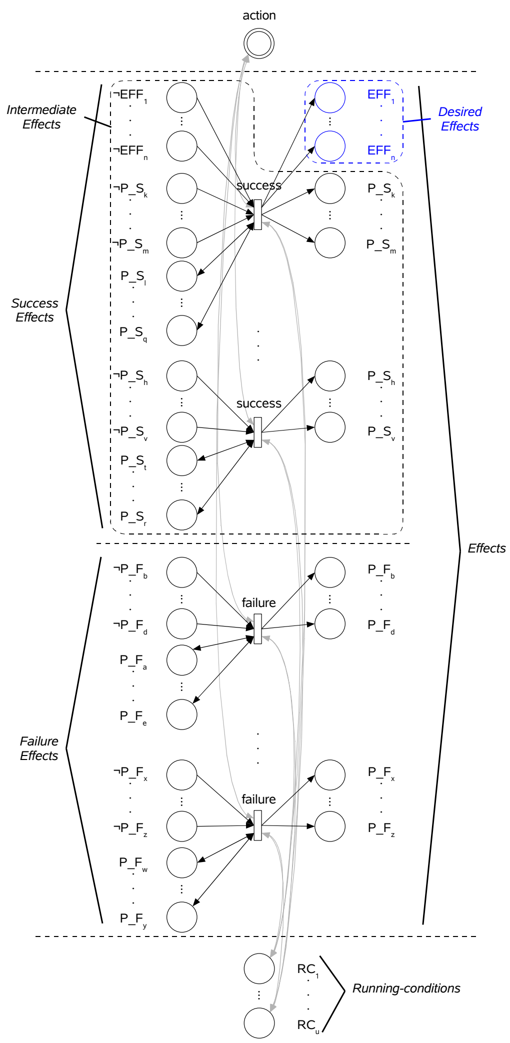

Fig. 2. General `action` model.

These models represent a never ending action, that when enabled e.g., by a higher layer, with the right conditions, increases the probability of certain effects to happen. Although the action can be always enabled, if the *running-conditions* are not met, it is as if the action were not enabled. This action model represents the actual action coded on the robot, and might be used to generate the action source code structure with the various conditions tests, leaving the user only to fill the method itself.

An action model for grabbing a ball, `grabBall`, can be seen in Fig. 3. The purpose of the `grabBall` action is to grab the ball, thus, when executed, it increases the probability of the predicate `HAS_BALL` (the only desired effect) becoming true, given that the ball can be seen and the robot is near the ball (running-conditions). The expected increase in the probability of success is reflected in the action model, by containing a stochastic transition from the place associated with the predicate `NOT_HAS_BALL` to the one associated with the predicate `HAS_BALL`.

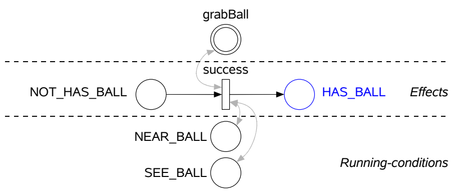

Fig. 3. Model of the `grabBall` action.

The example depicted in Fig. 3, includes no undesired effects explicitly modelled, however, these still might happen. For analysis, the models are composed with the Environment model, thus including different transitions, although with much lower probability of occurrence, and not specifically associated with the given action. Thus, undesired action outcomes are included only when they are relevant to the action being modelled, i.e., the probability of their occurrence is far higher due to the modelled action being executed. Note that the *running-conditions* might be an empty set, meaning the respective action can be executed with success at any time.

The *action place* will be used by a higher level to control the execution of an action, and to represent the action itself. The *running-conditions* and *effects* can be used to determine when an action can and should be used, respectively.

## C. THE ACTION COORDINATOR

The Action Coordinator layer holds the task plans used to execute tasks, consisting of a network of actions. In its most basic form, we can build a plan using the GSPN formalism and the action models.

*1) THE ACTION MODEL:* From the definition of the action Petri net model at the Action Executor level, we extract important details for the selection of actions:

- **Pre-conditions**: conditions that should be met when starting the action to ensure its usefulness in a given context;
- **Running-conditions**: conditions that must be met during the action execution for it to have any impact on the environment;
- **Success-conditions**: conditions that are satisfied when the action is successful.

*Definition 3.6:* The full action model, at the Action Coordinator level, is a GSPN $\langle P, T, I, O, \mathcal{M}_0, R, S\rangle$ automatically obtained from the action model at the Action Executor level $\langle \mathtt{P}, \mathtt{T}, \mathtt{I}, \mathtt{O}, \mathtt{M}_0, \mathtt{R}, \mathtt{S}\rangle$, where:

1) $P = P_A \cup P_P \cup P_S \cup P_R$, where:
   - $P_A = \mathtt{P}_\mathtt{A} = \{a\}$ is the action place set;
   - $P_P = \mathtt{P}_\mathtt{R} \cup \neg\mathtt{P}_{\mathtt{E}_{\mathtt{S}_\mathtt{D}}}$ is the pre-conditions places set;
   - $P_S = \mathtt{P}_{\mathtt{E}_{\mathtt{S}_\mathtt{D}}}$ is the success-conditions places set;
   - $P_F = \neg\mathtt{P}_\mathtt{R}$ is the failure-conditions places set;
2) $T = T_S \cup T_{ES} \cup T_{EF}$, with $T_S \cap T_{ES} = \emptyset$, $T_S \cap T_{EF} = \emptyset$ and $T_{ES} \cap T_{EF} = \emptyset$, where:
   - $T_S = \{start\}$ contains one transition that enables the action, labeled *start*;
   - $T_{ES} = \{success\}$ contains one transition that ends the action with success, labeled *success*;
   - $T_{EF}$ is the set of the transitions that end the action with failure, which have the label *failure*;
3) $\forall_{t_j \in T} \forall_{p_n \in P \setminus P_A}, (i_{nj} = 1) \Rightarrow (o_{jn} = 1)$. If there is an arc from a place $p_n$, associated to predicate $P()$, to a transition $t_j$, then there is an arc from $t_j$ to place $p_n$;
4) $\forall_{p_n \in \mathtt{P}_\mathtt{P}}, (i_{nj} = 1) \wedge (o_{jn} = 1)$, where $t_j = start$. There is an input arc and an output arc from every pre-condition set place to transition *start*;
5) $\forall_{t_j \in T_S} o_{jn} = 1$, where $p_n \in P_A$. All the start transitions have an output arc to the action place;
6) $\forall_{t_j \in (T_{ES} \cup T_{EF})} i_{nj} = 1$, where $p_n \in P_A$. All the end transitions have an input arc from the action place;
7) $\forall_{p_n \in P_F} \exists_{t_j \in T_{EF}}, (i_{nj} = 1) \wedge (o_{jn} = 1)$. There is a failure transition for each running-condition, with an input and output arc from the corresponding running-condition;
8) $\forall_{p_n \in \mathtt{P}_\mathtt{S}}, (i_{nj} = 1) \wedge (o_{jn} = 1)$, where $t_j = success$. There is an input and output arc from every success-condition set place to transition *success*;

*Lemma 3.3:* Item 3 from Definition 3.6 implies that when a token is consumed from a predicate place, it is placed again on the same place, which means that the high-level action model does not violate the predicate places definition and associated lemma.

The general model of the action at the Action Coordinator level, corresponding to the model in Fig. 2 at the Action Executor level, is depicted in Fig. 4. As an example, the Action Coordinator layer model of the action `grabBall`, corresponding to the Action Executor layer model seen in Fig. 3, is depicted in Fig. 5.

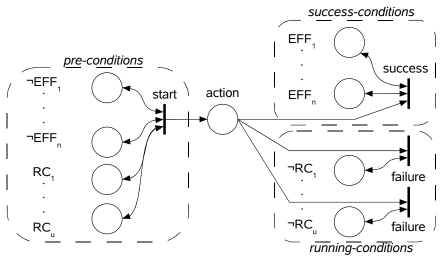

Fig. 4. General model for the action at the Action Coordinator level.

Both the models in Figs. 4 and 5 are complete models, representing an abstraction from the lower level action models, which ensure maximum efficiency, since the actions can only be enabled if they can change the environment, i.e., if the running-conditions are met and the associated successful effects are not already caused on the environment. The simplest model at the Action Coordinator level consists only of the action place, which corresponds to selecting an action independently of its usefulness in the given context.

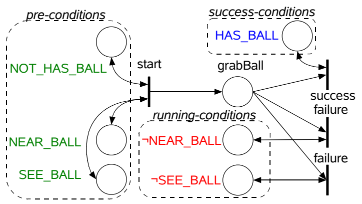

Fig. 5. Model of the `grabBall` action at the Action Coordinator level.

*2) THE TASK PLAN:* A full task plan model can be obtained by systematic plugging of actions, using information retrieved from the action models, either by a human or by a planning algorithm. These task plans run directly on the robot, so the user does not have to write any actual code.

As an example, in Fig. 6 (left) is a GSPN model of a simple Petri net task plan of a SCORE GOAL task in the robotic soccer domain, which includes four possible actions (`standBy`, `getBall`, `takeBall2Goal` and `shoot2Goal`) chosen according to the actions *pre-conditions*, *running-conditions* and *success-conditions*. This plan was manually specified and we did not always use all the information available when enabling an action, creating a simple, but not optimal, plan. An automatically generated optimal plan would use the full action models in order to guarantee modularity and optimal action selection.

In this example we considered a *simple* `shoot2Goal` action that could only be successful near the opponent's goal. As another example, we considered a *full* `shoot2Goal` action which can score from everywhere, but includes failures, so as to have higher scoring probability nearer the opponent's goal (see Fig. 6 on the right). Here, a random switch is included, so that when the robot is running `getBall` and grabs the ball, both $t_3$ and $t_5$ are enabled. Changing the probability associated with each of these transitions changes how often, under the above conditions, the `takeBall2Goal` or the `shoot2Goal` actions are selected. Moreover, these could be changed at runtime. For instance, if a game is near the end and our team is losing, we might prefer to shoot to the goal more often, while if it is winning, we might prefer to stay on the safe side and only shoot if the probability of scoring is very high, i.e., near the opponent's goal.

## IV. RESULTS ON THE ANALYSIS OF ROBOTIC TASKS USING PETRI NETS

Analysing a given Petri net based task plan consists of composing all the models together, building a single Petri net, and performing analysis on that Petri net. This composition is performed based on the premise that every place with the same label is in fact the same place, and each transition is different, regardless of its label. Analysing the resulting GSPN, equivalent to a semi-Markov process, we can obtain

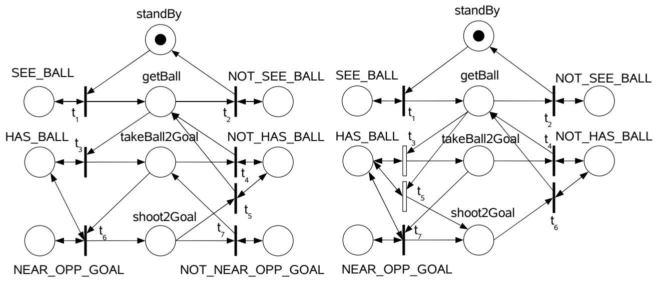

Fig. 6. Two SCORE GOAL Petri net task models: a simple one (left) and more complex one, with random switches (right).

various results, such as: existence of deadlocks, probability of reaching a given state, time to reach a given state.

### A. THE SETUP

The results were obtained using a robotic soccer scenario, with the goal being to score in the opponents goal. We used several different models to test the influence of the realism of the actions and environment models. We also tested the impact on goal achievement with different task plans.

The environment model included the ball position model depicted in Fig. 1 plus a similar robot position model. The ball position model was slightly changed in order not to allow the ball to leave a goal once inside. The used actions were `standBy`, `getBall`, `takeBall2Goal` and `shoot2Goal`. We considered two different environment models: the *Probabilistic Environment* and the *Deterministic Environment*. The difference is that the *Deterministic Environment* did not contain stochastic transitions, thus not considering external changes in the environment.

As explained in Section III-B, the action models are always stochastic timed, even if they only contain success transitions (in which case the failures appear only by composition of the models). To test the impact on modelling the action failures we considered the two `shoot2Goal` actions described earlier: the *simple* `shoot2Goal` and *full* `shoot2Goal`.

Finally, to test the impact of action selection, we considered four different task plans. The first case consists in using the task plan depicted in Fig. 6 (left). The other three cases correspond to using the task plan depicted in Fig. 6 (right) with different probabilities associated with the random switch, i.e., `takeBall2Goal` is chosen with probability: *a)* 1, *b)* 0.5 and *c)* 0.

In each test, the ball was initially placed near our own goal and the robot in the midfield. Furthermore, we considered that the robot could always see the ball and changed the action models to prevent the robot from entering the goals. The rates used in the models were (in *time units*) $1/1$ for success transitions, $1/4$ for failure transitions, $1/10$ for environment stochastic transitions and, $1/4$ and $1/6$ for scoring from the midfield and near our own goal, respectively.

### B. RESULTS

Given the different models, we tested eight setups using the TimeNET tool [2], obtaining the results shown in Table

TABLE I
RESULTS

| | Env. | Actions | $\mathbf{P_{Score}}$ | $\mathbf{T_{1\%}}$ | $\mathbf{T_{50\%}}$ | $\mathbf{T_{90\%}}$ |
|---|---|---|---|---|---|---|
| 1 | D | S | 1 | 1.3 | 4.8 | 8.1 |
| 2 | P | S | 0.75 | 1.2 | 6.0 | - |
| 3 | D | $F^a$ | 1 | 1.3 | 5.7 | 13.2 |
| 4 | P | $F^a$ | 0.68 | 1.2 | 9.5 | - |
| 5 | D | $F^b$ | 1 | 1.1 | 6.5 | 15.5 |
| 6 | P | $F^b$ | 0.68 | 1.0 | 9.9 | - |
| 7 | D | $F^c$ | 1 | 0.9 | 7.3 | 18.0 |
| 8 | P | $F^c$ | 0.66 | 0.83 | 10.7 | - |

I. *P* and *D* on the *Environment* column denote using the *Probabilistic* or *Deterministic* model respectively. In the *Actions* column, *S* and *F* mean using the *simple* and *full* `shoot2goal` action respectively. The superscript *a*, *b* and *c* refer to the three task plan models described previously.

In Table I, $\mathbf{P_{Score}}$ is the highest score probability achievable, while $\mathbf{T_{n\%}}$ is the time taken to achieve $n\%$ scoring probability. All the time values are in *time units*.

Qualitatively, we concluded there is always the possibility of a deadlock, corresponding to a goal score, since the ball cannot leave the goal, and is not reachable inside the goal.

The results show that, like in reality, since the robot is not the only responsible for changing the environment, the probability of scoring a goal is not 1 when the environment model is stochastic. Furthermore, the more complete the models are, the lower is the probability of scoring, and the longer it takes to achieve high scoring probability.

Comparing the values obtained with $F^a$, $F^b$ and $F^a$, an interesting, but expected result appears: when we choose to shoot not only from near the opponent's goal, the $1\%$ scoring probability is achieved more quickly, however, it takes longer to achieve high scoring probability, i.e., the long run performance is decreased.

## V. CONCLUSIONS AND FUTURE WORK

### A. Conclusions

Petri nets provide a practical and intuitive way of modelling robotic tasks and associated components, being also appropriate to monitor the execution of tasks given their graphical nature. The fact that a GSPN is equivalent to a Markov chain brings an additional advantage by allowing the use of currently available tools and techniques in order to extract important *a priori* information about a given task.

Being able to model the actions more thoroughly at a lower level allows for mores realistic models. We are simultaneously able to create an abstract and simpler model of the action to be used at higher levels, without compromising the analysis possibilities. Furthermore we can create all the models separately and build the task plan by creating a network of actions. This task plan can be ran directly on the robots for execution purposes, and for analysis purposes we compose all the models that were designed separately onto one Petri net, and analyse that net.

### B. Future Work

We have recently finished implementing a Petri net executor in our MeRMaID architecture [12]. We are now starting to use the referred models and analysis tools in real world experiments, and plan to have further results of those soon.

We plan to work further on composition operators, used to compose actions into behaviours, and study the possibility of improving analysis time by simplification of the tasks by replacing networks of actions by behaviours. These can also have an important part in cooperative behaviours.

Another improvement that is planned is the use of data obtained from real experiments to verify and complete the various Petri net models, particularly the rates used in the stochastic timed transitions.

As mentioned in Section III-C.2, the task plan can be obtained by using planning algorithms. The information given by the definition of the actions at the Action Coordinator level should suffice in order to use algorithms for deterministic planning, while the information given by the definition of the actions at the Action Executor level could be used for probabilistic planning. The Markov chain obtained by the composition of all the models can be used to compute the optimal action selection for any given state.

## VI. ACKNOWLEDGEMENTS

This work was supported by Fundação para a Ciência e Tecnologia (FCT) under grant SFRH/BD/12707/2003, and by ISR/IST pluriannual funding from FCT through the POS_Conhecimento Program that includes FEDER funds.

## REFERENCES

[1] N. Viswanadham, Y. Narahari, *Performance Modelling of Automated Manufacturing Systems*, Prentice Hall, 1992.

[2] A. Zimmermann, *A Software Tool for the Performability Evaluation with Stochastic Petri Nets*, http://pdv.cs.tu-berlin.de/ timenet/, 2001.

[3] A.Dominguez-Brito, M. Andersson, H. Christensen, *A Software Architecture for Programming Robotic Systems based on the Discrete Event System Paradigm*, Report CVAP244, ISRN KTH/NA/P–00/13–SE, Centre for Autonomous Systems, KTH (Royal Institute of Technology), 2000.

[4] J. Kosecka, H. Christensen, R. Bajcsy, *Experiments in Behaviour Composition*, Robotics and Autonomous Systems, vol. 19, pp. 287-298, 1997.

[5] B. Espiau, K. Kapellos, M. Jourdan, D. Simon, *On the Validation of Robotics Control Systems Part I: High Level Specification and Formal Verification*, Report 2719, INRIA, 1995.

[6] B. Damas, and P. Lima, *Stochastic Discrete Event Model of a Multi-Robot Team Playing an Adversarial Game*, Proc. of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles, 2004.

[7] C. Cassandras, S. Lafortune. *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 1999.

[8] M. Andersen, R. Jensen, T. Bak, M. Quottrup, *Motion Planning in Multi-Robot Systems using Timed Automata*, Proc. of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles, 2004.

[9] L. Montano, F. García, J. Villaroel, *Using the Time Petri Net Formalism for Specification, Validation, and Code Generation in Robot-Control Applications*, The International Journal of Robotics Research, vol. 19, no. 1, pp. 59-76, 2000.

[10] V. Ziparo, L. Iocchi, *Petri Net Plans*, Proc. of the Fourth International Workshop on Modelling of Objects, Components, and Agents, 2006.

[11] D. Milutinovic, P. Lima, *Petri Net Models of Robotic Tasks*, Proc. of the IEEE International Conference on Robotics and Automation, 2002.

[12] N. Ramos, M. Barbosa, P. Lima, *MeRMaID - Multiple-Robot Middleware for Intelligent Decision-making*, Proc. of IAV2007 - 6th IFAC Symposium on Intelligent Autonomous Vehicles, 2007.