Robotic Tasks Modeling and Analysis Based on Petri Nets*

Hugo Costelha and Pedro Lima
Instituto de Sistemas e Robótica
Instituto Superior Técnico
Av. Rovisco Pais, 1, 1049-001 Lisboa, Portugal
{*hcostelha,pal*}*@isr.ist.utl.pt*

***Abstract*— This paper proposes a method of defining and analysing robotic tasks using Petri Nets. Both the robot behaviors and environment are modelled using Generalized Stochastic Petri Nets (GSPNs). Each action is modelled separately and composed with others to provide a complete task execution. The use of Petri Nets allows the qualitative and quantitative analysis of the task execution.**

***Index Terms*— DEDS, Robotic Tasks, Modeling, Analysis.**

I. Introduction

The number of tasks accomplished by robots is highly increasing and with it, are the variety and complexity of the tasks that a robot needs/should be able to perform. One needs to have formal methods of specifying, controlling and analysing a task, specially in hazardous environments or very high risk missions where one cannot afford for unexpected results to happend.

Most of the work found on the literature concerning the design of robotic tasks using Discrete Event Systems (DES) is based on Finite State Automata (FSA) for code generation [2], qualitative specifications [1] and some quantitative specifications [4]. Reference [1] also provides a high modularity in tasks specification.

Recent work has also been done in multi-robot analysis and modeling using FSA ([6] and [7]).

Some work using Petri Nets to design robotic tasks under some temporal requirements, focusing also on the generation of real-time, error-free code can be found in [3].

Part of the work presented in this paper is based on the ideas introduced in [6] and [5], where in [5] a framework for qualitative and quantitative performance using Petri Nets is introduced.

The need for a modular design of complex robotic tasks with some *a priori* knowledge of quantitative and qualitative properties of the designed task is the main motivationfor this work. Petri Nets come up has the perfect modeling and analysis tool to acomplish that task. It's modeling and analysis power applied to a single or multi robot task definition can prove to be quite effective. The fact that Petri Nets and computer programs that run in the robotic platforms are very tightly coupled, makes the implementation of behaviors and the monitoring of those behaviors the appropriate environment for the application of Petri Nets.

*This work is partially supported by FCT Grant SFRH/BD/12707/2003 to H. Costelha

II. Petri Nets Models

In this work we use Petri Nets to model, design and evaluate a robotic task exectution.

One of the reasons of choosing Petri Nets over FSA was that combining Petri Nets does not lead to the same state explosion as combining FSA. Of course when performing the analysis the problems still exists, but at least visually we don't need to loose information and can easily monitor the all system running using the original Petri Nets models in an easier way. It also seems more apropriate in a robotic environment to use a Petri Net approach, to model each resource separetly and have the state of the robot has the state of all its components.

The choosen environment is robotic soccer and we model the behavior execution of a soccer robot in a single robot environment. We define a set of behaviors that can be used to perform a robotic task under supervisor control. We modeled only three behaviors, enough to model an atack behavior of a soccer robot without increasing the visual complexity of the system. It's clear however that the system can be easily extended and that even a multi-robot environment can be analysed (although we only focus on a single-robot environment).

We wish to analyse a robotic task in two perspectives: quantitative and qualitative. For that we will use respectively marked Petri Nets and General Stochastic Petri Nets (GSPNs) as explained in Sec. III-A and Sec. III-B.

The composition of the various Petri Nets is done using place composition, so take into account that when looking at places with the same label, they are in fact the same place. The same doesn't happen with the transitions, as they are all different (of course that doesn't mean that we cannot incur in some simplifications concerning possible duplicate transitions with the same inputs and outputs).

The places labels in the Petri Net models concerning the robots actions and sensors have a trailing `PN_` where `N` is the robot's number. This does not remove any generality to the models has they can be used in any other robot by simply changing the robot's number. The existence of the number in the labels will be important in the future when considering multi-robot models and the need to control each robot's actions independently.

To simplify the visual complexity of the Petri Nets depicted in the following figures, whenever we have a place and a transition connected with two arcs in opposite directions, we draw a single arc with arrows in both ends.

A. The Environment

To be able to analyse the tasks in simulation, we have to model the environment, the robot's sensors response to environment changes and the response of the environment to the robots actions. While these models might not have a direct relation with the program that will eventually run on the robot, their modeling is very important in simulation results.

We divided the field in six regions: `OwnGoal`, `OwnSide`, `OpSide`, `OpGoal` and two other regions, `OwnGoalScored` and `OpGoalScored`, which respresent a goal scored in each goal (these regions are only available to the ball's position and not the robot's position).

We assumed that both the ball and the robot cannot go from one region to the other without going through intermediary regions.

1) The Ball Position Model and the Player Position Model: For now we will make some assumptions relative to the ball movement and possible positions. We will consider that the ball can only be moved by a robot, so we have a place for each possible ball position (six places) and no transitions between them (these will appear from the robots actions models).

The player position model is very similar with the ball position model, apart from the `GoalScored` regions. In this case too, the transitions between the robot's possible positions will appear from the robot's actions models.

If we wanted to model the ball movement, or the possibility of someone picking up the robot and change its position, we would add transitions connecting the various places, representing those movements.

2) The Sensors Models: Considering the possible robot and ball positions one can model the sensors of the robots. We consider as sensors all the relevant information that is extracted from the real sensors on a robotic platform. In the case presented in this paper we consider the following sensors: SEE_BALL/N_SEE_BALL, HAS_BALL/N_HAS_BALL, NEAR_BALL/N_NEAR_BALL, NEAR_OP_GOAL, GOAL_SCORED.

We will not consider errors or uncertainty on the sensors, only on the actions effects on the environment, thus NEAR_OP_GOAL and GOAL_SCORED are in fact the same as the player's position OP_GOAL and the ball's position OP_GOAL_SCORED.

The HAS_BALL resource model is presented in Fig. 1 has an example of a resource model. In general we model separately the transition from the the *resource available* to *resource not available* and from *resource not available* to *resource available*, has is the case of the model shown in Fig. 1, where only the model of the transition from P1_HAS_BALL to P1_N_HAS_BALL is shown.

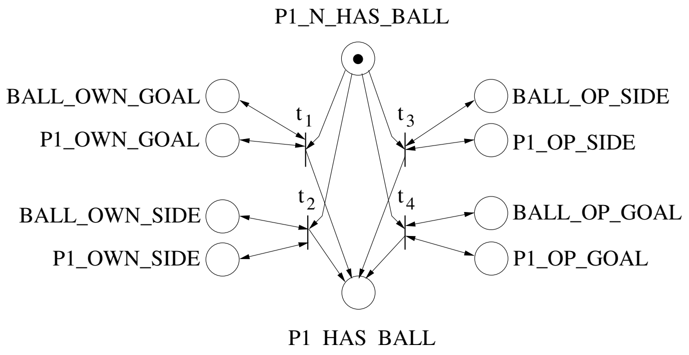

Fig. 1. HAS_BALL resource model for Player 1.

The resources models should be always immediate transitions since they are basically the verification of environment conditions.

B. The Robotic Behavior

The Robotic Behavior represents the basic module of the task execution. The composition of the various behaviors will ultimately form the Robotic Task. These are tightly coupled with the program that runs in the robot and can also be used to generate code (one of the planned future features). It will also allow to monitor the task performing by looking at the tokens traveling through the Petri Net.

The Robotic Behavior has a standard form consisting of inputs, outputs, pre-conditions , post-conditions and the behavior action. The pre-conditions and the post-conditions represent sensorial information and determin respectively the conditions to start and end the behavior. The inputs are used by the supervisor to control the behavior execution, while the outputs are used to keep knowledge of the behavior success or failure. For this work we started with three possible behaviors: `Move2Ball`, `TakeBall2Goal` and `Shoot`. This three behaviors allow to perform a complete soccer attack play and many different situations to emerge.

In Fig. 2 is depicted the `TakeBall2Goal` behavior. In this model we see the common structure that appears in all the behaviors, except for the *Pre-conditions*, the *Post-conditions* and the *Action*, which differ from behavior to behavior. The `TB2G_Action` place has a dashed line only to indicate that this place represents the core execution of the `TakeBall2Goal` robotic behavior: the robotic action. This is still a higher level representation of the model, and at this level one cannot see what the action really does. From Fig. 2 it also gets clear how the pre-conditions and post-conditions influence the behavior execution.

The *Action*, in it's simplest form, can be just a movement (e.g. go forward) but it can also be a more complex action, represented by a more complex Petri Net (e.g. rotate to the ball and follow it). It is in the *Action* Petri Net that we will model the robot's actions plus the uncertainty and time issues concerning their execution.

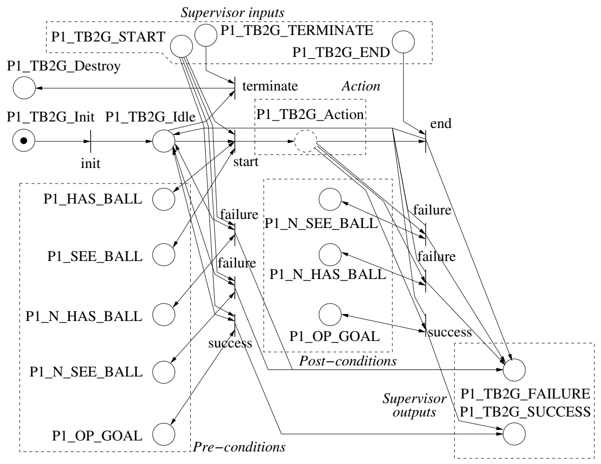

Fig. 2. `TakeBall2Goal` behavior.

The model of the *TakeBall2Goal Action* associated with the place `TB2G_Action` can be seen in Fig. 3. In this action, the uncertainty introduced reflects the fact that the robot might not be able to proceed do the next field region. It's rather simple to introduce other uncertainties/failures (e.g. the ball going to another region but the robot not moving to another region) but we decided to keep it simple for this case. On a successfull sequence of transitions we have the robot plus the ball going from it's starting region to the `OP_GOAL` region. Here also the time taken to take the ball from one region to the other is considered by the use of the stochastic timed transitions. The action Petri Net was not depicted in the behavior figure so that the behavior Petri Net could be more readable and to show how the various modules can be designed apart from each other for later integration.

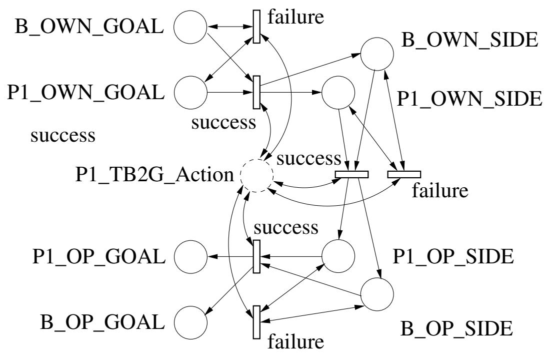

Fig. 3. `TakeBall2Goal` action.

It is clear now that a behavior starts when there is a "supervisor command" to start it, and that it stops upon failure/success of the pre-conditions and/or post-conditions or there is a "supervisor command" to end the behavior.

C. The Robotic Task

Having the robotic behaviors, the robotic task is simply an execution of those behaviors. If there weren't any uncertainties or failures, the execution of the task of scoring a goal would simply be the sequential execution of the behaviors: *Move2Ball* followed by *TakeBall2Goal* followed by *Shoot*. But we know that this will not happend always and in fact it would be the best scenario case in this task case.

In order to control the behaviors execution we need a supervisor . For this work we will use a very simple supervisor that the only thing it does is to allow the execution of any of the behaviors, but only one at a time. To make the supervisor a little more efficient, we use the starting pre-conditions of each behavior as a condition to be met in order for that behavior to be selectable. Thus we get the supervisor depicted in Fig. 4

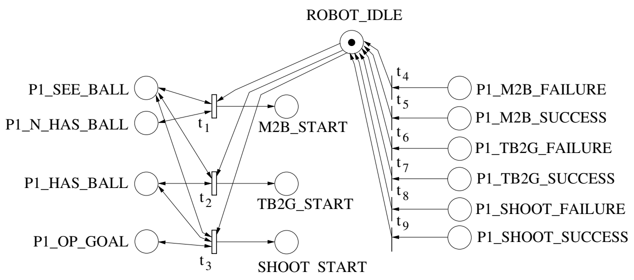

Fig. 4. Supervisor for the Score Goal Task.

The overall task model results from the composition by places of all the models previously addressed.

III. Task Analysis

The main purpose of the modeling using the Petri Nets is the available analysis and the various properties that can be studied. In this work we try to do both types of analysis: quantitative and qualitative.

With the complete task model we can now study it's various properties as can be seen in the following sections. One interesting point of the method depicted in this paper is that, besides analysing the task itself, one can also analyse each behavior independentely (by having the supervisor enabling only the behavior in question). This might give us some ideas on problematic behaviors that need to be improved and allows the tunning of a behavior before inserting it in the behaviors set for real use.

A. Qualitative Analysis

To perform the qualitative analysis we consider all the transitions in the task Petri Net as immediate transitions, using the general form of the Petri Nets. We can then use several techniques such as Coverability Tree or Structural Analysis to determin the following properties: Boundedness, Safety, Blocking, Coverability, Conservation and Liveness.

B. Quantitative Analysis

It is in the performance analyses that the GSPNs of the task plays its role. The modeling of the task using GSPNs allows us to analyse the task as a Markov Decision Problem and to take advantage of all the existing Markov Decision

Theory to determine several quantitative properties (such as minimum time to score a goal, best action selection in order to achieve that minimum time, steady state analysis, etc.).

The task's Petri Net (GSPN model) has an equivalent Continuous Time Markov Chain (CTMC) where the state is defined as a vector containing the number of tokens in each place (see [8]) and an associated finite cost that depends only on the active state and the control action selected from a set of possible control actions. The transition probabilities from one state to another depend only on the active state and the selected control action. Then by uniformization of this CTMC we get a Discrete Time Markov Chain (DTMC) and we have our Markov Decision Problem.

In this works case, the control policy is directly related with the supervisor's ”decisions” i.e., it will determin the best behavior selection at each state in order to achieve a determined goal.

For instance, if we want to minimize the time to score a goal, one can simply add a zero cost to the state(s) corresponding to a scored goal and a positive cost to every other state. The policy resulting from this minimization problem will be the optimal policy for scoring a goal given an initial state.

Other interesting analyses one can do in this process is for instance the time spent in the *TakeBall2Goal* behavior, the recurrence time of the *Shoot* behavior, and others. This can be done by working with the properties of the places belonging to the behaviors models. Considering the example of the time spent on the *TakeBall2Goal* behavior, it would be equal to the sum of the time spent on every state that contains at least one token on any place that belongs to the *TakeBall2Goal* behavior. This results can then be used to understand possible problems in our task design, and when applied to a real robot, it can be used not just to improve the real task execution, but also the model itself by comparing the real results with the simulated ones. This will ultimately lead to the design of a task with a very high *a priori* knowledge of the task execution results.

IV. Conclusions and Future Work

In the immediate future we expect to improve the action models (possibly adding in a first phase the possibility of using uncontrollable transitions and, on a second phase, unobservable transitions) and extend the available behaviors in order to apply the method with a real robot.

We will also extend the modeling and analysis capabilities to a multi-robot environment, specially considering that we we can have teammates and opponents playing in the environment and that one can't always know/predict the adversaries actions.

References

[1] J. Kosecka, H. Christensen and R. Bajcsy, ”Experiments in behaviour composition”, Robotics and Autonomous Systems, vol. 19, pp. 287-298, March 1997.

[2] A. Dominguez-Brito, M. Andersson and H. Christensen, ”A software architecture for programming robotic systems based on the discrete event system paradigm”, Tech. Rep. CVAP244, ISRN KTH/NA/P-00/13-SE, September 2000.

[3] L. Montano, F. Garcia and J. Villaroel, ”Using the time petri net formalism for specification, validation and code generation in robot-control applications”, in The International Journal of Robotics Research, vol. 19, pp. 59-76, January 2000.

[4] B. Espiau, K. Kapellos, M. Jourdan and D. Simon, ”On the validation of robotics control systems part I: High level specification and formal verification”, Tech. Rep. 2719, February 1995.

[5] D. Milutinovic and P. Lima, ”Petri net models of robotic tasks”, In Proc. of IEEE 2002 Int. Conf. on Robotics and Automation (ICRA 2002), May 2002.

[6] B. Damas and P. Lima, ”Stochastic discrete event model of a multi-robot team playing an adversarial game”, 5th IFAC/EURON Symposium On Intelligent Autonomous Vehicles, July 2004.

[7] M. Andersen, R. Jensen, T. Bak and M. Quottrup, ”Motion planning in multi-robot systems using timed automata”, 5th IFAC/EURON Symposium On Intelligent Autonomous Vehicles, July 2004.

[8] C. Cassandras and S. Lafortune, ”Introduction to Discrete Event Systems”, Kluwer Academic Publishers, 1999.