Analysing Robot Movement Using the Sensoria Methods[*]

Maurice H. ter Beek[1], Alessandro Lapadula[2], Michele Loreti[2], and Claudio Palasciano[3]

[1] ISTI–CNR, Pisa, Italy
terbeek@isti.cnr.it
[2] DSI, Università di Firenze, Italy
{lapadula,loreti}@dsi.unifi.it
[3] MIP Politecnico di Milano, Italy
palasciano@mip.polimi.it

Abstract. In this paper, we give a recount of the application of Sensoria approaches, languages, and tools to the modeling of movement of the robot that has taken the lead role in Sensoria demonstrations at the exhibitions ICT 2008 in Lyon and FET 2009 in Prague. The demos were centred around a robot-bowling game that actively involved the visitors in programming a robot that plays bowling, using some of the techniques developed in Sensoria in order to predict the outcomes of the game according to their design choices. Specifically, the Sensoria techniques have been used for the analysis of functional and non-functional properties of the system, both in the ex-post analysis of the robot movement during the demo and in the ex-ante analysis of the possible robot configurations during the design of the robot and of the demo itself. This paper presents how the techniques have been applied and to what extent the results of the application match the real robot behavior. The Sensoria modeling and analysis techniques used are the UML4SOA graphical modeling language, the Performance Evaluation Process Algebra PEPA, the UMC model checker and the Markovian process algebra MarCaSPiS.

1 Introduction to the Bowling Robot Case Study

The Bowling Robot case study was at first developed as a robot-bowling game scenario for demonstrating Sensoria's software engineering approach and tools in a practical way during hands-on demonstration, after which it evolved into a case study, apt to show Sensoria results applied to the design and development of a real system, the robot that plays bowling itself. Specifically, the Bowling Robot has been used as a showcase for the modeling techniques and tools, in particular qualitative and quantitative modeling, and was not intended to address service composition/orchestration issues.

In fact, initially, the Bowling Robot scenario was developed in order to show in advance to the demo participants, who were asked to make some choices during the programming of the Robot Player, how different control policies and

[*] This work has been partially sponsored by the project Sensoria, IST-2005-016004.

parameter choices may affect the robot's performance. This was aimed at allowing the demo participants to reason on the robot's parameters and environment and make sensible choices in order to try to win the robot-bowling game. In this phase, the Robot Player development was meant to be conducted in a traditional spiral approach including design, construction of the robot, program writing, and testing 'on the field'. We would have designed the Robot Player first and then applied SENSORIA results in order to be able to show how SENSORIA would support the visitor in programming the robot during the demo.

During the design of the robot, some complex and interleaved choices were needed concerning the mechanical features, control policies, and possible parameter sets. During testing experiments on the field, unexpected properties of the system under development came out and the necessity to analyze the scenario in more detail appeared clear, in order to ensure proper behavior of the robot while allowing a visitor at the same time to enjoy a nice choice among different control policies and parameter values. Some 'hidden' needs were identified, such as 'the robot must always complete the proper actions and arrive at the end of the lane'. We therefore expressed as design questions desirable properties of the system. During the development of the robot very soon the testing of the overall behavior of the robot program was substituted by experiments planned for the determination of the environment and the robot's 'internal' parameters such as the robot speed. This led us to focus on the usage of different SENSORIA languages and tools, not only during the demo sessions but during the design of the demo itself and, finally, to further analysis after the demo development ended. This process may also be considered as a hint of how a traditional system development approach may change with the availability of SENSORIA.

This paper is organized as follows. In Sect. 2 we describe the context of the robot-bowling game scenario, after which we describe the Bowling Robot case study and its relevance to SENSORIA in Sect. 3. The results of the application of four SENSORIA techniques to the case study are presented in Sect. 4, while we report our conclusions and the lessons learned in Sect. 5.

2 Context Description

The Bowling Robot case study context is a game scenario developed for demonstrating SENSORIA's potential impacts on the software systems development process in a SOA context presented at the ICT exhibit in Lyon, November 2008, and at the FET exhibit in Prague, April 2009.

The Bowling Robot demo scenario is focused on people who want to play bowling on the Internet by means of a so-called 'Virtual Bowling' service. The Virtual Bowling service provides an actual competition performed by a robot, called "Player". The game is personalized according to each virtual player's profile (e.g. her/his gender, depending on whether a red or blue ball is used). This information is provided to the Player by a second robot, called "Coach", that receives from the Virtual Bowling service the gamers' requests including information about the virtual player and communicates them to the Robot Player.

In the Bowling Robot scenario, we imagine that a human player, i.e. one of the exhibit visitors, is to specify the program that determines the behavior of the Robot Player during the game. The Bowling Robot case study is centred around this concept: the visitor, in order to optimize the robot's performance, has to reach the maximum score in the game taking into account the physical characteristics of the robot and of the environment, and in this process s/he is supported by the Sensoria tools and languages. In particular, Sensoria modeling approaches facilitate the prediction of robot behavior at design time.

The Bowling Robot demo presented at ICT'08 and FET'09 has four phases:

phase 1 The visitor programs the robot using the standard Lego GUI.
phase 2 The visitor is able to make choices in order to improve her/his Lego program with the support of Sensoria analysis techniques.
phase 3 The visitor has access to the Virtual Bowling service that activates the bowling-robot game.
phase 4 After the game, a score is computed for each visitor in order to award a daily Sensoria robot-bowling champion cup.

3 Case Study Description

In the robot-bowling game, the main characteristic of the environment is the floor of the bowling alley, that presents a pattern with a radial geometry, darkening from white to black (see Fig. 1): the maximum black point is in the optimal launching position (2), right in front of the first pin.

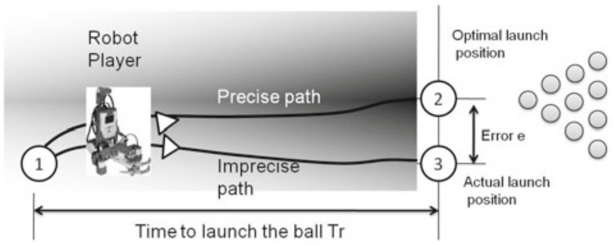


Fig. 1. Bowling Robot scenario

In this context, the Robot Player begins from loading position (1) at the start of the bowling lane (80 cm) and aims to reach the optimal launching position (2). The Robot Player detects the color of the floor by means of two light sensors. The Player, aiming to follow the maximum value of the gradient towards the black, at fixed time intervals T measures the level of white/black on the floor and decides to go forward or turn to correct its path. Finally, due to possible imperfect path finding, the Player may reach the end of the lane in position (3) and launch the ball (actually, drop it on a ramp towards the pins). The Player recognizes the end of the lane as it is marked by a silver ribbon giving the maximum white value on the floor MAX. The computed game score depends on:

1. the number of pins knocked down,
2. the time to arrive at the launching position and drop the ball (Tr), and
3. the distance from the optimal launching position, i.e. error e.

Accordingly, the visitor is asked to program the Robot Player to pull down as many pins as possible in the shortest possible time with the maximum precision (minimize e). For the sake of simplicity, we define as 'precise launches' those with error $e < 10$ cm, evidenced by a green zone 20 cm wide at the end of the lane, centered w.r.t. the optimal launching position. The programming choices available to the visitor are taken from a limited set, in order to allow each visitor to complete the demo in a few minutes.

Having fixed the path dimensions and geometry we decided the mechanical structure of the Player that includes two sensors to measure the color of the path on the floor, two wheels, two motors, and, in front, a simple mechanical hand (served by a third motor) that allows to get and release the ball. Concerning movements, a Player can go forward or turn approximately on its place. In particular, each of the two wheels of the Player is connected to a motor that is powered with a fixed power level during the game. The Player can go forward or turn by switching on and off only one of the two motors (e.g. in order to turn right, the Player switches off the right motor while the left motor is on). In Fig. 2 the Robot Player of ICT'08, based on standard Lego NXT robot kit, is shown.

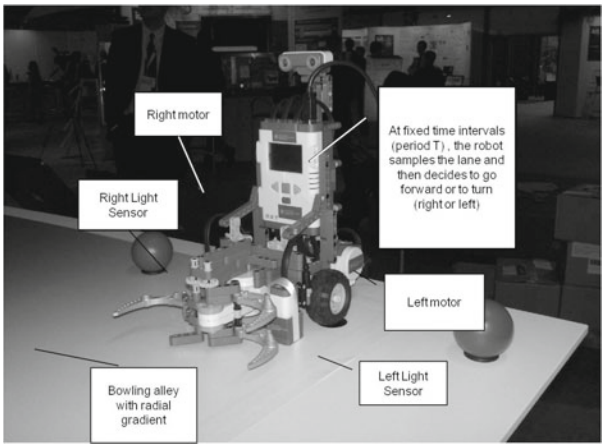

Right motor
Right Light Sensor
At fixed time intervals (period T), the robot samples the lane and then decides to go forward or to turn (right or left)
Left motor
Left Light Sensor
Bowling alley with radial gradient

Fig. 2. The Robot Player

In order to assess its position on the path, the robot has two light sensors aimed at the floor, one on each side near each wheel. The sensor readings are numerical values that are higher as the color of the path goes from white to black. The measured radial gradient is 2.66 sensor units/cm.

At fixed time intervals the Robot Player stops, reads the sensors and then, by comparing the readings with a threshold value S, decides to go forward or turn (right or left) for that time interval T. The time interval can be imagined as a sampling period or alternatively its reciprocal is the frequency at which the system is controlled. Finally, when the Player reads maximum value of light reading (MAX), s/he decides that the end of the alley is reached and launches the ball. The overall workflow includes, after reading the sensor values, computing a specific value D depending on the sensors readings, which is to be used afterwards to decide whether the robot is on the correct path (and has to go forward) or not (and has to turn), and in the latter case, in which direction to turn.

During the Lego programming phase of the demo, the visitor builds up the Lego program by completing a pre-defined program flow:

1. choose blocks (tagged with self-explicating titles) from a predefined set and
2. set some program parameters within the program blocks.

During this phase the visitor is asked to choose between two different "Follow the Gradient" strategies, represented by two specific personalized program blocks, and to specify how long is the light sensors' sampling period T, among the set 0.2, 0.5, 0.7 and 1.0 seconds.

The two possible "Follow the Gradient" strategies available to the programmer of the Player are described in Figs. 3 and 4. The first (Strategy #1, Fig. 3) takes into account that, given the geometry on the floor path and the position of the two light sensors (symmetrically positioned with reference to the longitudinal axis of the Player), the difference between the left and right sensor readings should be approximately zero if the robot points towards the optimal launching position (2). Accordingly, at each step, the robot computes the difference D and the sign *SIGN* between the readings of the right and the left sensor. If the difference D is lower than a specified threshold value S, then the Player goes forward, i.e. the robot is 'following the gradient'. Otherwise, the robot turns right or left depending on the sign of the computed difference D, in order to correct the path.

The second strategy (Strategy #2, Fig. 4) stems from the following consideration: if the robot is correctly 'following the gradient' towards the maximum black point (2), the difference of the reading of one sensor with the reading of the same sensor at the previous time interval should be positive and greater than the threshold value S. In this case we imagine that each of the sensors is controlled by one concurrent task in the program and checks this difference to decide whether or not to turn. Each of the tasks, consequently, as in Fig. 4, executes the main loop that includes stop, reading, compute, decide, and go forward/turn depending on the decision for that interval.

In detail, refering to the task that reads the left sensor, after having read the sensors and computed the difference $leftDelta(t)$ between the current sensor reading $leftData(t)$ and the previous one $leftData(t-1)$, the robot decides to turn if the difference is below the threshold value S and to go forward otherwise. In particular, it can be shown that if the robot aims to the left of the optimal launching position (2), then the $rightDelta(t)$ difference is higher than

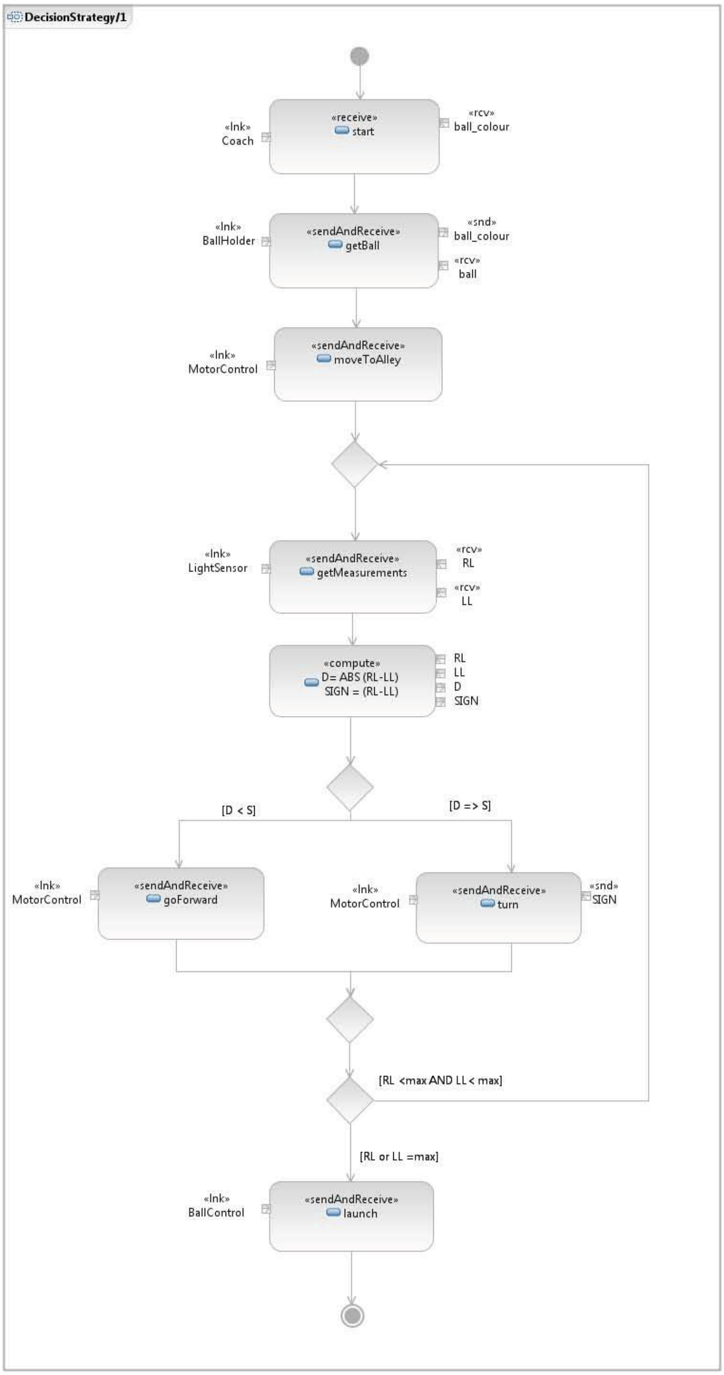


Fig. 3. "Follow the gradient" – Strategy #1

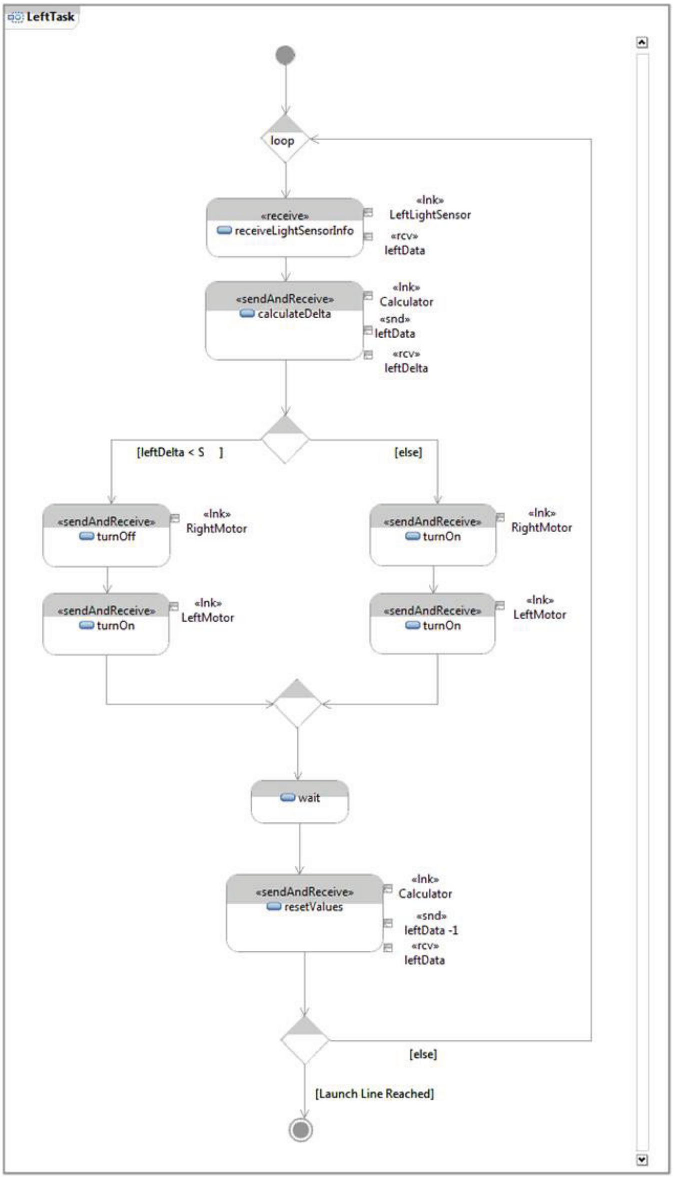


Fig. 4. "Follow the gradient" – Strategy #2, left task

the *leftDelta*(t) difference and therefore the robot has to turn to the right; vice versa if the robot aims to the right of the optimal launching position (2). Consequently, to achieve this behavior, as explained in Sect. 2, with Strategy #2, if the *leftDelta*(t) difference is below the threshold S the robot has to turn right, i.e. it has to turn off the right motor, otherwise the robot switches on both motors. As we will show later, Strategy #2 as presented so far is flawed: it has been devised on purpose in order to allow a positive search for programming errors.

To summarize, sampling period T and threshold value S are the main parameters that can be chosen to determine the robot's behavior and affect its performance, i.e. the time to launch the ball (Tr), the error in reaching the optimal position (e) and, ultimately, the number of knocked down pins. According to that, during the design of the Robot Player, concerning most of all design decisions and in particular the definition of the possible "Follow the Gradient" strategies and of the possible set of parameters T and S, the following 'design questions' arose, more or less explicitly:

DQ1 Does the robot always arrive at the end of the lane, and launch the ball?
DQ2 Does the robot mostly launch the ball within approximately one minute?
DQ3 What is the robot's performance according to different parameter choices? Are there interesting/unexpected situations that SENSORIA might evidence?

4 Use of SENSORIA Tools and Results

As shown in the previous sections, SENSORIA has been used both during the Robot Player design and development and during the demo. The Bowling Robot scenario has allowed the use of the following SENSORIA tools and languages:

– For qualitative analysis, error identification: WS-Engineer/LTSA (see Chapter 4-4) and the UML profile UML4SOA (see Chapter 1-1).
– For qualitative analysis, system properties verification: UML model checker UMC and its logic UCTL (see Chapters 4-2 and 4-3).
– For quantitative analysis, non-functional properties: Performance Evaluation Process Algebra PEPA with IPC compiler (see Chapter 5-4 and references).
– For qualitative/quantitative analysis: both functional and non-functional properties: MarCaSPiS, Markovian extension of the process calculus CaSPiS (see Chapters 2-1 and 5-1).

The first two analyses, performed with UML4SOA and UMC models, allowed us to reply to **DQ1**. The quantitative analysis with PEPA allowed us to ensure robot behavior during the design and to show the visitor during the demo both the answer to **DQ2** and the robot's possible performance depending on different choices of design parameters, specifically parameter T (**DQ3**). Finally, analysis with MarCaSPiS allowed us to reply to **DQ1–3** using a more detailed model of the Robot Player behavior. The SENSORIA models have been developed in each specific language as a mirror of the Lego program deployed on the robot.

4.1 Qualitative Analysis, Error Identification (UML4SOA Model)

During the first design activities of the robot we used the WS-Engineer tool to check the Lego programs under development for errors. We used it to perform a qualitative analysis since, being based on LTSA, it allows model checking to be carried out on the UML4SOA model of the robot and to detect deadlocks and verify arbitrary properties stated in the process calculus FSP (see Chapter 4-4 for details). Specifically, the model was written in UML4SOA, which was transformed by means of the Sensoria MDD4SOA transformation tool (see Chapter 1-1) to generate BPEL code, which was then used as input for WS-Engineer that, in turn, internally converted the BPEL code into FSP.

The following property has been analyzed: if a task starts interacting with motors, both calls to each motor are executed without interruption. This allowed us to tell that Strategy #1 has no error, while Strategy #2 does have one. This error, specifically, is related to an undetermined behavior: as both the left and right task at the same time switch on and off the same robot motor (see Fig. 5), the resulting behavior is that the Player is not able even to move. This allowed the visitors to avoid using the flawed Strategy #2. It was possible to show to the demo visitors that the Player under Strategy #2 would go back and forth on her/his place without moving properly, but only one of them asked to do so.

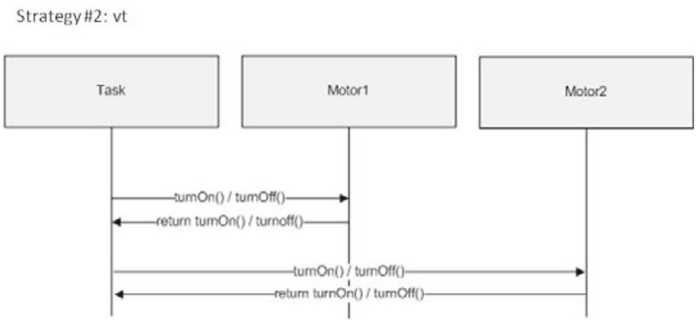


Fig. 5. Violation trace for Strategy #2

4.2 Qualitative Analysis, System Properties Verification (UMC/UCTL Model)

An alternative technique for qualitative analysis is offered by model checking with UMC, which allows the formal verification of the dynamic behavior of UML models. A UMC model of the robot's behavior consists of a description in UML state machines. The robot's desirable properties are then expressed using the UCTL logic, which is essentially the full modal/propositional μ-calculus extended with higher-level CTL/ACTL-like operators and structured action expressions (see Chapters 4-2 and 4-3 for details), and verified with UMC.

UMC enables modeling a Robot Player as a composition of evolving and communicating objects, where objects are class instances (Coach, Light Sensor, Robot Player, etc.), which together perform the desired action sequence. The set of objects and classes which constitute a system can be described in UML by a structure diagram, while the dynamic behavior of the objects can be described by associating a UML statechart diagram to their classes. Each object of the system will therefore behave like a state machine. An excerpt (class Robot) of the UMC model of the Player under Strategy #1 is as follows.

```
Class Robot is
 Signals:
  //signals received from DecisionStrategy component
  start;
  getBall;
  moveToAlley;
  getMeasurements;
  doComputation;
  turn;
  goForward;
  launch;
  //Signals to/from other components
  ballColor(bc:Token);                      //reception of ball color
  receiveBall(b);                           //reception of ball
  moveDone;                                 //effected movement signal
  lightVal(rl_val: Token, ll_val: Token);  //reception of light values
 Vars:
  ball_color: Token := null;
  ball: Token := null;
  RL: Token := null;
  LL: Token := null;
  sign:Token := null;
  coachObj: Coach;
  ballHolderObj: BallHolder;
  motorControlObj: MotorControl;
  lightSensorObj: LightSensor;
  ballControlObj: BallControl;
  decisionStrategyObj: DecisionStrategy;
 State top = r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13
 Transitions:
  r0 -> r1 {- / decisionStrategyObj.nextAction(self)} //start request
  r1 -> r2 {start / coachObj.requestBallColor(self)} //receives decision strategy and
  r2 -> r3 {ballColor(bc) / ball_color := bc;        //then asks coach for ball color
           decisionStrategyObj.nextAction(self)} //receives ball color
  r3 -> r4 {getBall / ballHolderObj.requestBall(self,ball_color)} //asks ball holder for ball
  r4 -> r5 {receiveBall(b) / ball := b; decisionStrategyObj.nextAction(self)} //receives ball
  r5 -> r6 {moveToAlley / motorControlObj.requestMoveToAlley(self)} //motors to move on alley
  r6 -> r7 {moveDone / decisionStrategyObj.nextAction(self)} //receives movement done signal
  r7 -> r8 {getMeasurements / lightSensorObj.requestLightVal(self)} //asks for light values
  r8 -> r9 {lightVal(rl_val,ll_val) / RL := rl_val; LL := ll_val;
           decisionStrategyObj.nextAction(self)} //receives light values      //computes:
  r9 -> r10 {doComputation / decisionStrategyObj.calcDeltaAndSign(self,LL,RL)} //delta & sign
  r10 -> r11 {goForward / motorControlObj.requestMoveForward(self)} //motor to move forward
  r10 -> r11 {turn / motorControlObj.requestTurn(self)} //motor to turn //first get moveDone,
  r11 -> r12 {moveDone / decisionStrategyObj.decideLaunch(self,LL,RL)} //then launch decision
  r12 -> r8 {getMeasurements / lightSensorObj.requestLightVal(self)} //cycles to r8 for
  r12 -> r13 {launch / ballControlObj.launch} //launches ball        //new measurements
end Robot;
```

The analysis performed shows that the Robot Player under Strategy #1 is able to complete the actions required for a bowling game, i.e. the sequence of movements up to launching the ball while moving along the alley. Specifically, the following properties (written in UCTL) have been verified true for Strategy #1.

1. The robot always eventually launches the ball: $AF\ EX\ \{launch\}\ true$
   If this property holds, the system is not effected by deadlock in any possible trace before reaching a state in which the *launch* action can be performed.

2. The next action after a movement request is notifying movement completion:

$$AF\ EX\ \{requestMoveForward\ OR\ requestTurn\ OR\ requestMoveToAlley\}\ AX\ \{moveDone\}\ true$$

   If this property holds, the robot performs no other actions after each movement request, until the motor announcing the requested maneuver finished.
3. At least once, the proper action sequence for a bowling game is performed:

$$AF\ EX\ \{start\}\ AF\ EX\ \{getBall\}\ AF\ EX\ \{moveToAlley\}\ AF\ EX\ \{getMeasurements\}\ AF\ (EX\ \{goForward\}\ true\ AND\ EX\ \{turn\}\ true)\ AF\ EX\ \{launch\}\ true$$

   The fact that this property holds guarantees that the robot's behavior includes at least once the desired sequence of actions for a bowling game: starting movement, getting the ball, moving to the alley, get measurements and going forward (turning) and, finally, launching the ball.

During the demo we showed that the Robot Player's behavior under Strategy #1 followed the proper sequence of movements up until launching the ball.

4.3 Quantitative Analysis, Non-functional Properties (PEPA Model)

During the Player development a series of experiments had been performed in order to ensure proper behavior of the robot during the demo over a possible set of different design features, such as threshold value, distance between sensors, sampling period. The robot showed satisfactory behavior under the following parameters choices: distance between sensors $d = 9$ cm, threshold value $S = 20$ sensor units and sampling period in the interval from 0.2 to 1 sec. In particular, rather counter-intuitively, the robot behavior experimentally showed that, as the sampling period increases (e.g. from 0.2 sec to 1 sec) the time to roll the ball Tr decreases, i.e. the robot becomes quicker to reach the end of the alley while the frequency at which the system is controlled decreases (see Table 1).

Table 1. Experimental results

Sampling period T (sec)	Time to roll ball Tr (sec)	Distance from optimal launching position = error e (cm)	Precise occurrence (percentage of precise events)
0.2	53	4.3	100%
0.5	24	3.9	100%
0.7	22	6.7	80%
1.0	21	10.7	50%

In fact, the set of experiments performed (approximately 20 samples for each of the sampling period values) reported an overall average Tr of 53 sec for $T = 0.2$

and of 21 sec for $T = 1$ sec. The same experiments reported that the error e (distance from the optimal position (1)) increases from 4 cm to approximately 11 cm on average if the sampling period goes from 0.2 to 1: from this point of view, as it is most likely to be expected, the robot is more imprecise if the frequency at which the system is controlled decreases. In particular, the robot is most likely to be imprecise than precise if T is higher than 1 sec.

This was exactly what we needed for the sake of a game competition: the visitor is asked to choose the value of a control parameter (T) which is related to the trade-off between two performance indicators, i.e. time to launch Tr and precision (related to error e). This leaves her/him freedom to decide, as the information available is not enough to precisely forecast the outcome of their choice on the value of T (they were not allowed to know the rating calculation formula). To show *i*) on the one hand to the visitors how SENSORIA can predict the outcome of programmer's choices concerning the value of sampling period T and *ii*) on the other hand give an answer to the aforementioned design question **DQ2** concerning the overall time to roll the ball Tr, and specifically to the above mentioned unexpected behavior, we developed the following approximate model of the Robot Player behavior using the PEPA modeling approach (see Fig. 6).

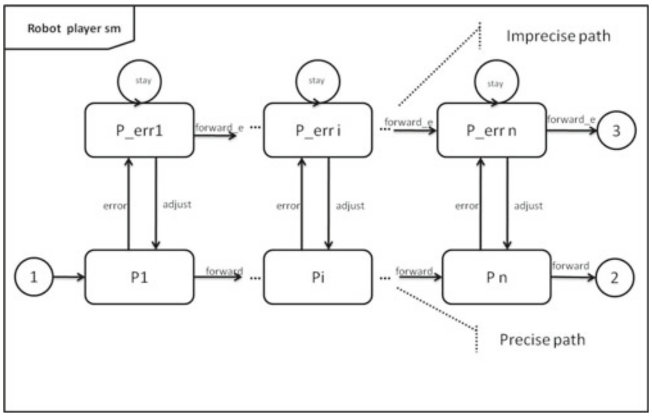


Fig. 6. State machine model of Robot Player

Concurrent systems can be modeled in PEPA as composition of components which undertake actions. In PEPA actions have a duration. Thus, the expression $(\alpha, r).P$ denotes a component which can undertake an action α at a rate r, to evolve into component P. The rate r models a delay of variable duration. Delays are samples from an exponential random variable with parameter r. See Chapter 5-4 and its references for details on PEPA.

We intuitively describe the PEPA model using a state machine model (see Fig. 6). The robot starts its run on the bowling alley in state $P1$ (we did not model the straight line movement from starting position (1) to the beginning of the bowling alley) and completes the run in two possible ways: it follows the 'precise' path indicated by the states from P1 to P n or the imprecise path, related to the states from P_err1 to P_err n. Due to the geometry of the problem, we imagine that after each step the robot is in one of two possible states: pointed towards the position (2) or not. Accordingly, the first situation corresponds to a state P on a precise path and the second to the robot on an imprecise path.

The robot completes its path to the end of the alley (80 cm long) in a total number of n steps, each one corresponding to a single sampling cycle. At each step i, with reference to Fig. 7, if the robot is on the precise path, it can deviate from the optimal path with probability e_prob, going in state P_err i, or, with probability $(1 - \mathit{e\_prob})$, can assess that it has to go forward straight to position (2) (optimal route) so it goes in state P i+1 on the precise path. The probability *e_prob* is related to the error that affects the readings of the light sensors (and most likely to mechanical structure imperfection and differences between motors as well). If the robot is on the imprecise path, it has three possibilities: to remain onto the imprecise path without progressing towards the end of the alley (with probability *a_prob*), to remain onto the imprecise path while anyway progressing towards the end of the lane (with probability *fe_prob*, or, finally, to go back to the precise path (i.e. to point to the maximum black point) with probability $1 - \mathit{a\_prob} - \mathit{fe\_prob}$. We completed the model defining the rate at which the system goes from state to state on the precise and imprecise paths, *forwardrate*, related to the speed of the robot.

To perform the PEPA analysis, we observed the robot behavior to assess the values of probabilities *e_prob* and *fe_prob*, and measured the robot movements during a single sampling cycle to assess rate *forwardrate*. To estimate *e_prob*, for each value of sampling period T, we counted how many times the robot moved from a precise to an imprecise path, i.e. if pointing to the optimal lauch position the Player did not go straight. Likewise, for each T, to estimate the probability of remaining on the imprecise path (*fe_prob*) we counted how many times the Player remained in the wrong direction if s/he headed in the wrong direction. According to experiments and analysis done on Strategy #1 deadlock-free characteristics, we decided to consider *a_prob* null (i.e. the robot always progresses towards the end of the alley). Concerning the rate *forwardrate*, we measured both the linear and the angular speed of the robot, measuring respectively the distance (cm) and the rotation (degrees) over several sampling cycles.

Table 2. Internal robot parameters

Sampling period T (sec)	Distance covered (DC) in a single sampling cycle (cm)	Rotation during sampling period (degrees)	Rotation (radians)	Sampling rate ($1/T$)	# sampling cycles to cover alley (80/DC)
0.2	0.4	2.9	0.05	5.0	200
0.5	2.4	16.6	0.29	2.0	36
0.7	3.4	24.5	0.43	1.43	23
1.0	5.5	46.7	0.81	1.0	14

Table 2 shows, for each sampling period T, the rectilinear distance covered (DC) by the robot during a single sampling cycle, the rotation during sampling period (radians), the corresponding rate at which sampling occurs ($1/T$) and the number of sampling cycles that are needed for the robot to cover the 80 cm distance to the end of the alley. According to this measurement we defined the parameters of the Player PEPA model (see the code that follows below).

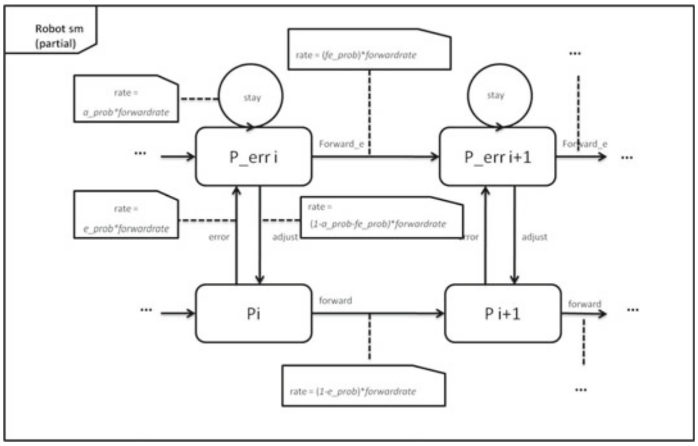

Robot sm (partial)
rate = (fe_prob)*forwardrate
rate = a_prob*forwardrate
stay
stay
P_err i
Forward_e
P_err i+1
Forward_e
rate = e_prob*forwardrate
rate = (1-a_prob-fe_prob)*forwardrate
error
adjust
error
adjust
Pi
forward
P i+1
forward
rate = (1-e_prob)*forwardrate

Fig. 7. State machine model of Robot Player: action rates

```
// PEPA model of Robot Player for T = 1 (forwardrate = 1)
n = 14;
// 4 error paths taken if robot is on right path in 49 movements
e_prob = 0.082;
a_prob = 0.00;
// 48 error paths taken if robot is on error path in 49 samples
fe_prob = 0.98;
forwardrate = 1.0;

Step = (forward,infty).StepDone + (start,infty).Step;
StepDone = (stop1,infty).Step + (start, infty).Step;
Steps = Step[n][stop1,start];
Step_e = (forward_e,infty).StepDone_e + (start,infty).Step_e;
StepDone_e = (stop2,infty).Step_e + (start,infty).Step_e;
Steps_e = Step_e[n][stop2,start];
AllSteps = (Steps < > Steps_e);

PlayerStop1 = (start,1.0).Player;
PlayerStop2 = (start,1.0).Player;
Player = (forward,(1-e_prob) * forwardrate).Player              // Precise path
       + (error,e_prob * forwardrate).PlayerError
       + (stop1,(1-e_prob) * forwardrate).PlayerStop1;
PlayerError = (adjust,(1-a_prob-fe_prob) * forwardrate).Player   // Imprecise path
            + (forward_e,fe_prob*forwardrate).PlayerError
            + (stop2,fe_prob * forwardrate).PlayerStop2;
Player < forward,forward_e,start,stop1,stop2 > AllSteps         // System equation
```

In particular, to take into account the effective robot speed, the PEPA calculations have been performed for each T value varying in accordance with the number of steps n of the model. To simplify the modeling, we considered only the linear speed: as one can see in Table 2, for $T = 0.2$ sec, first line of the table, if the robot covers DC = 0.4 cm in each step, it needs $n = 200 = 80/0.4$ steps.

It must be said that as the robot deviates from the original straight line with high probability, this approximation leads to computing a total time to launch the ball Tr that is certainly underestimated. Anyway, we had also to take into account the fact that the computation time is a polynomial function of the number n of steps in the model[1], therefore we had to take it as low as possible. Moreover, specifically in the case of $n = 200$ steps, the available computing resources have been not enough to perform the computations, even with that approximation; in fact, we reduced the model to 20 steps, introducing further approximations for the sake of the demo, that we will not discuss here.

We are mainly interested in an intuitive explanation of the results obtained with SENSORIA modeling techniques compared with the experimental findings. The results of PEPA's quantitative analysis are shown in Fig. 8: the cumulative probability functions of the time to roll the ball Tr for the robot ending in the precise path (line labeled stop_1) and for the robot ending out of the precise 'zone' along the imprecise path (line labeled stop_2). The left graph shows Tr for sampling period equal to 0.5 sec; that on the right for sampling period $T = 1$ sec.

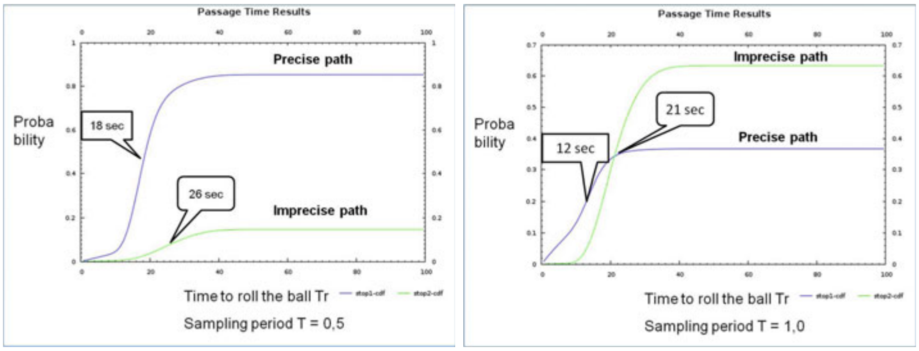


Fig. 8. Cumulative probability distributions of time to launch the ball (Tr)

The graphs show that with sampling period $T = 0.5$ sec the robot is precise in approximately 84% of the cases and on average launches the ball in $Tr = 18$ sec, while in 16% of the cases it is imprecise and launches in $Tr = 26$ sec. Our PEPA model computes $Tr = 20$ sec to roll the ball on average on all launches (precise or imprecise). If the sampling period is $T = 1$ sec the robot is precise in 37% of the cases and on average launches in 12 sec, while it is imprecise in approximately 63% of the cases and launches in 21 sec. The total average Tr is 18 sec.

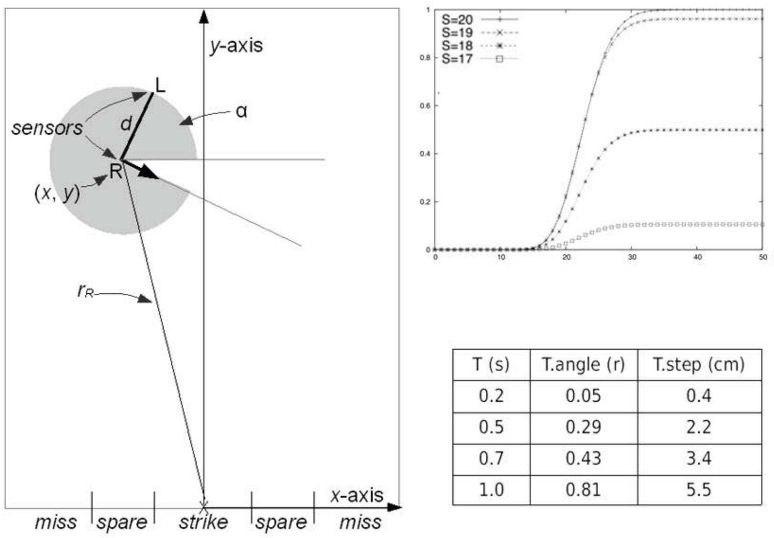
[1] Computational resources increase with space state size, that, in this PEPA model, is proportional to the number of steps n squared.

We can say that the results of the PEPA quantitative analysis intuitively agree with the experiments performed with the real robot. As shown in Table 3, both data series (computed and experimental) with increasing sampling period T, show that time to roll the ball Tr decreases, while the robot is less precise and becomes most likely imprecise if $T = 1$.

Table 3. Comparison of results of PEPA analysis and experimental data

T (sec)	PEPA average time Tr (sec)	PEPA precision (% precise launches)	Experimental average Tr (sec)	Experimental precision (% precise launches)
0.5	20	84%	24	100%
0.7	19	63%	22	80%
1.0	18	37%	21	50%

To summarize, the PEPA quantitative analysis, even if based on an approximate model, has allowed us, while choosing a sensible set of parameter values for the sampling period T according to all design choices made before, to explain the reasons of an unexpected behavior of the system during the system design and development. This led us to decide that the sampling period T, that was related to the unexpected behavior, was the one to be chosen as a programming parameter for the demo. Furthermore, the same analysis have been used to show the demo participants how the robot behavior could be predicted.



T (s)	T.angle (r)	T.step (cm)
0.2	0.05	0.4
0.5	0.29	2.2
0.7	0.43	3.4
1.0	0.81	5.5

Fig. 9. Some details of the Bowling Robot scenario in MarCaSPiS (left + table) and results of experiments with T equal to 0.5, 0.7, and 1.0 sec (right)

4.4 Qualitative/Quantitative Analysis, Functional and Non-functional Properties (MarCaSPiS Model)

Both qualitative and quantitative aspects of the Bowling Robot scenario are analysed with MarCaSPiS, the stochastic extension of CaSPiS. The analysis concerns the mathematical modeling of the robot behavior in accordance with Strategy #1. The analysis was performed by taking into account the environmental parameters (like the geometry of the path gradient), the internal robot parameters (like robot speed and sensor reading errors) and the programming parameters available in the design (sampling period T and threshold value S).

Figure 9 shows some details of the scenario modeled in MarCaSPiS. Its main parameters are the Cartesian coordinates of the right-sensor $R = (x, y)$ and the angular coordinate of the robot's direction α. These parameters univocally identify the robot's position. Angular and rectilinear movements, denoted by $T.angle$ and $T.step$ respectively, can be defined as functions of the parameter T (see, e.g., values in Fig. 9 for sampling periods 0.5, 0.7, and 1 sec). No MarCaSPiS analysis was performed for $T = 0.2$ sec as the model's state space size exceeded the available computation resources. The Cartesian coordinates of the left-sensor can be obtained from R and α by using the following trigonometric functions:

$$L = (x_L, y_L) = (x - d \cdot \sin(\alpha), y + d \cdot \cos(\alpha))$$

A robot decides upon the next action according to the floor gradient. An approximation of the floor gradient is modeled by means of function $L(r)$ defined as $MAX + k \cdot r$, where MAX is the maximum light sensor value, constant k is -2.66 unit/cm and r is one of the following right/left sensor radial coordinates:

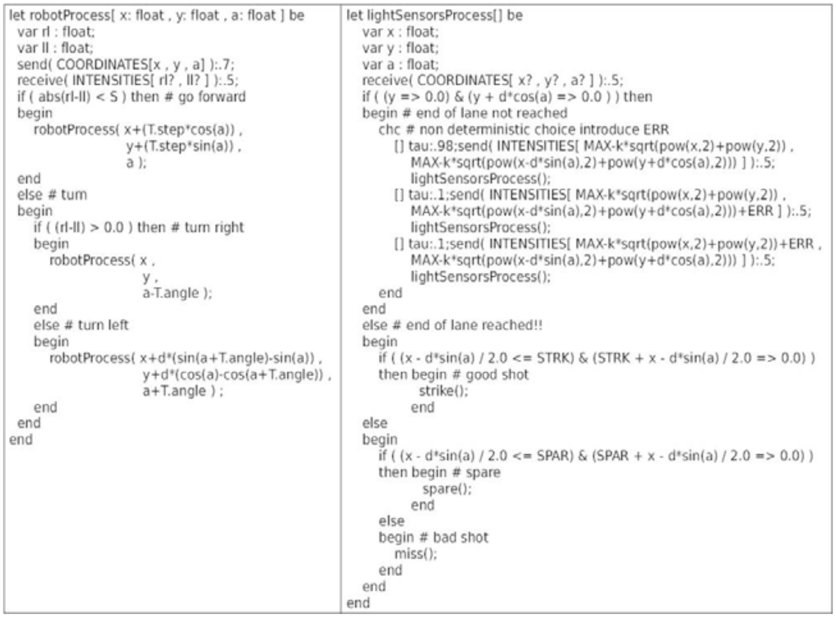
$$r_L = \sqrt(x_L^2 + y_L^2) \qquad\qquad r_R = \sqrt(x^2 + y^2)$$

The end of the bowling lane (along the x-axis) is partitioned into three zones w.r.t. the outcomes of a launch: all pins are knocked down (strike), pins are knocked down but some pins are missing (spare), and all pins are missing (miss).

The Robot Player is modeled in MarCaSPiS as a process that invokes a light sensor service. The robot-side interaction protocol is modeled by means of process "robotProcess" described in Fig. 10 (left column). This process sends the robot position to process "lightSensorsProcess" to receive the light-intensity measurements. Measurements permit deciding whether to move forward or turn left or right, or whether or not to launch the ball. This last decision is made when the robot reaches the end of the lane (see also process "lightSensorsProcess").

After receiving measurements rl and ll, process "robotProcess" compares the difference $abs(rl - ll)$ with the value of the threshold parameter S to determine the new robot position. After each movement, the new robot position (denoted by R' and α') is calculated by using the following trigonometric functions:

turn right $R' = (x, y)$ and $\alpha' = \alpha - T.angle$
turn left $R' = (x + d \cdot (\sin(\alpha') - \sin(\alpha)), y + d \cdot (\cos(\alpha) - \cos(\alpha')))$ and $\alpha' = \alpha + T.angle$
go forward $R' = (x + T.step \cdot \cos(\alpha), y + T.step \cdot \sin(\alpha))$ and $\alpha' = \alpha$



```
let robotProcess[ x: float , y: float , a: float ] be
  var rl : float;
  var ll : float;
  send( COORDINATES[x , y , a] ):.7;
  receive( INTENSITIES[ rl? , ll? ] ):.5;
  if ( abs(rl-ll) < S ) then # go forward
  begin
     robotProcess( x+(T.step*cos(a)) ,
                   y+(T.step*sin(a)) ,
                   a );
  end
  else # turn
  begin
     if ( (rl-ll) > 0.0 ) then # turn right
     begin
        robotProcess( x ,
                      y ,
                      a-T.angle );
     end
     else # turn left
     begin
        robotProcess( x+d*(sin(a+T.angle)-sin(a)) ,
                      y+d*(cos(a)-cos(a+T.angle)) ,
                      a+T.angle ) ;
     end
  end
end
```

```
let lightSensorsProcess[] be
   var x : float;
   var y : float;
   var a : float;
   receive( COORDINATES[ x? , y? , a? ] ):.5;
   if ( (y => 0.0) & (y + d*cos(a) => 0.0 ) ) then
   begin # end of lane not reached
      chc # non deterministic choice introduce ERR
         [] tau:.98;send( INTENSITIES[ MAX-k*sqrt(pow(x,2)+pow(y,2)) ,
            MAX-k*sqrt(pow(x-d*sin(a),2)+pow(y+d*cos(a),2))) ] ):.5;
            lightSensorsProcess();
         [] tau:.1;send( INTENSITIES[ MAX-k*sqrt(pow(x,2)+pow(y,2)) ,
            MAX-k*sqrt(pow(x-d*sin(a),2)+pow(y+d*cos(a),2)))+ERR ] ):.5;
            lightSensorsProcess();
         [] tau:.1;send( INTENSITIES[ MAX-k*sqrt(pow(x,2)+pow(y,2))+ERR ,
            MAX-k*sqrt(pow(x-d*sin(a),2)+pow(y+d*cos(a),2))) ] ):.5;
            lightSensorsProcess();
      end
   end
   else # end of lane reached!!
   begin
      if ( (x - d*sin(a) / 2.0 <= STRK) & (STRK + x - d*sin(a) / 2.0 => 0.0) )
      then begin # good shot
            strike();
           end
   else
   begin
      if ( (x - d*sin(a) / 2.0 <= SPAR) & (SPAR + x - d*sin(a) / 2.0 => 0.0) )
      then begin # spare
            spare();
           end
      else
      begin # bad shot
         miss();
      end
   end
end
```

Fig. 10. MarCaSPiS interaction protocol

Errors can occur in the interaction protocol during light sensors reading, as modeled in Fig. 10 (right column); "lightSensorsProcess" introduces errors through two measurement types: a correct one (evaluated by function $L(r)$ defined above) and an incorrect one (one for each sensor) characterized by error *ERR*.

After receiving a new robot position, "lightSensorsProcess" controls if the launching line is reached by checking the y-coordinate of each sensor. The outcome of a launch is determined in terms of the robot-midpoint position by checking whether the x-coordinate $x - (d/2) \cdot \sin(\alpha)$ belongs to one of the zones strike, spare, or miss (characterized by constants *STRK* and *SPAR)*.

MarCaSPiS' operational semantics permits characterizing the stochastic behavior of systems, whose properties can be specified with SOSL (Service-Oriented Stochastic Logic, see Chapter 5-1) and automatically checked using SOSL–MC.

We are primarily interested in an intuitive explanation of the results by showing the feasibility of our methodology with MarCaSPiS. We report an analysis based on experiments performed using MarCaSPiS in comparison to experimental results. We focus on the probability of the set of computations that lead to a state satisfying a generic property φ. This can be evaluated by means of the SOSL formula *true* $\{*\}$ U φ. When specifying the formula, we use property φ = *"strike"* to identify states of the model which correspond to an outcome *"strike"*; we use properties φ = *"spare"* and φ = *"miss"* to identify states corresponding to outcomes *"spare"* and *"miss"*, respectively. Figures 9 (right) and 11 show graphics generated by SOSL–MC illustrating the results of three sample experiments. We assume that the Robot Player is already on the alley and ready to bowl. Initially, the robot position is given by $R = (0cm, 80cm)$ and $\alpha = \pi$. The common parameters used in all experiments are: $d = 9cm$ (distance between sensors), $MAX = 711unit$ (max-black intensity), $STRK = 10cm$ and $SPAR = STRK + 12.5cm$ (strike and spare zones), and error $ERR = 2unit$. To display the results, we simulate model computations in the time interval $[0, 50]$.

The first graphic of Fig. 9 (right) permits answering a more general formulation of **DQ2** regarding whether the robot launches the ball within k time units (for a given k occurring in $[0, 50]$) arriving at strike zones and spare or miss zones. The former type of zone corresponds to precise launching, the latter to imprecise launching. The used parameters are: sampling periods $T = 0.5$ sec and $T = 0.7$ sec, and threshold parameter $S = 20unit$. This experiment also permits answering a formulation of **DQ1** regarding whether and how a robot arrives at the end of the bowling lane. To comment on some of the results of Fig. 9 (right), it shows that a robot with sampling period $T = 0.5$ sec is more precise (arriving at strike zones) than one with parameter $T = 0.7$ sec (which reaches launching zones more quickly, but can arrive at spare or miss zones) and much more than one with parameter $T = 1$ sec. It furthermore shows that for $T = 0.5$ the robot is most likely to be precise, according to experimental data.

We can say that the results of the MarCaSPiS analysis are intuitively in agreement with the experiments performed with the real robot. As shown in Table 4, both data series (computed and experimental) with increasing sampling period T, show that time to roll the ball Tr decreases, while the robot is less precise and becomes most likely imprecise if $T = 1$.

Table 4. Comparison of results of MarCaSPiS analysis and experimental data

T (sec)	MarCaSPiS average time Tr (sec)	MarCaSPiS precision (% precise launches)	Experimental average Tr (sec)	Experimental precision (% precise launches)
0.5	23.34	100%	24	100%
0.7	20.74	95%	22	80%
1.0	20.67	85%	21	50%

The last experiments are designed to answer questions of type **DQ3** on how robot behaviors change depending on different model parameters. Properties φ are specified to denote *"strike"* or *"spare"* outcomes separately. Figure 11 illustrates the evolving of the probability of a robot launching the ball with parameters $T = 0.5$ sec w.r.t. variations of parameter S from $20unit$ to $17unit$. In particular, Fig. 11 (left) illustrates the decreasing probability of arriving at a strike zone by reducing the value of S; conversely, Fig. 11 (right) illustrates the increasing probability of arriving at a spare zone by reducing the value of S.

To conclude, our MarCaSPiS methodology provides both 'intuitive' and effective estimations of robot behaviors (w.r.t. variations of model parameters) and a formal basis to reason on other functional properties of the Robot Player.

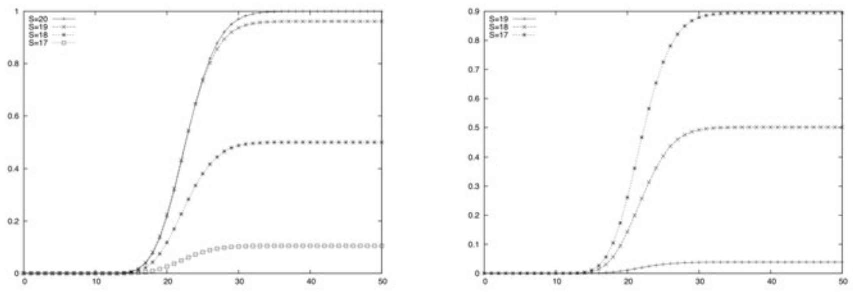


Fig. 11. Results of experiments with $\varphi =$ *"strike"* (left) and $\varphi =$ *"spare"* (right)

5 Conclusions and Lessons Learned

Concerning the demo, the Bowling Robot scenario has been particularly effective both in attracting visitors at the SENSORIA booths and in fostering their interest into the SENSORIA project after first impression. Nearly 200 visitors came to the SENSORIA booths during each exhibit and approximately 30 visitors each day participated in the robot-bowling game. Visitors judged particularly effective the way SENSORIA analysis (both qualitative and quantitative) helped making the choices on the program parameter values and policies, so that each demo session has been performed in a few minutes, as requested by the demo specifications, and time remained for the visitor to ask questions about SENSORIA.

During the development of the robot, on the one hand, very soon the testing of the overall behavior of the robot program was substituted by experiments planned for the determination of the environmental parameters, such as the linear and angular speed of the robot or the probability distribution of the sensor reading errors, showing a hint of how the SENSORIA approach, especially in a SOA concurrent environment, might lead to anticipate during the design phase a more detailed specification of the functional and non-functional aspects of the system and of its components. On the other hand, concerning the analysis with PEPA and MarCaSPiS, we discovered that, depending on the choice of system parameters, it is very easy to incur into a state-space explosion, i.e. the model dimensions grow to the extent that calculations are not possible due to limitation of the computing resources or take too much time for the project purposes or resources. In order to avoid this, it is important to carefully plan the experiments to be done and in particular the possible approximations that might be enacted.

Acknowledgments. This work benefited from discussions with P. Mayer (who wrote the UML4SOA model and did the WS-Engineer analysis) and A. Clark (who helped a lot in writing the PEPA model). Special thanks to A. Fantechi, who reviewed the paper, for his useful suggestions.