

Simultaneous Planning and Acting in Robotic Soccer

Andrés Bonilla

University of Puerto Rico, Mayagüez

andres.bonilla@upr.edu

ABSTRACT

Performing coordinated real-time actions in an uncertain evolving environment requires interpreting noisy probabilistic information about a situation and acting based on these interpretations. This task becomes much more complex when the environment contains other agents, some of which antagonize each other. RoboCup (The World Cup Robot Soccer) is an attempt to promote AI and robotics research by providing a common task for evaluation of various theories, algorithms, and agent architectures [5]. A robotic soccer simulator used in RoboCup competitions was chosen as a testbed for a simultaneous action/inference AI agent. The model is comprised of temporal Bayesian nets that maintain a probabilistic model of the environment, and Petri nets that control a procedural representation of action. Both components work in conjunction to model the agent's behavior; and communication is achieved with the simulator in a client-server fashion using UDP. The client was written using the Atan API, a set of Java classes for interfacing with the RoboCup Server. The goal of this work is to control simulated soccer players using an existing graphical Petri/Bayes net editor, and observe the quantitative performance of the nets in the editor and their qualitative performance in the behavior of the soccer agents.

1. INTRODUCTION

One difficult problem for modern Artificial Intelligence is to make useful, structured actions in real-time, uncertain environments. Real-world actions require complex structure in several forms, such as temporal structure, coordination between actions, and compositional structure [1]. For the most part, human behavior follows regular procedural patterns; every day we wake up, shower, brush our teeth, etc. The general goals of these procedures remain constant from day to day, but the lower level actions that compose the procedures are never the same; they are influenced by a dynamic environment which we must observe and make inferences on. If one day we turn the shower on and feel that the water is scalding hot, we don't jump in to the shower as normal, because we infer there is a high probability of getting burned if we do. This kind of inference composes a real-time belief network which this work will model using a graphical representation of probability distributions called Bayesian networks.

Actions as simple as picking up a toothbrush can be broken down into more atomic muscle movements that are never executed in the exact same way, but in general work to address the same routine task. This work will model quasi-atomic parametric soccer actions such as dashing and shooting using a set of petri nets that are fed inferences about the game by a set of temporal Bayesian belief networks.

Temporal Bayesian nets are suitable for environment representation because the world often contains hidden state, from which we receive only noisy samples [1]. The agent's view of the world is composed of Bayesian belief nets, which change over time in reaction to observed probabilities, and according to a set of programmed conditional probabilities. In this case the RoboCup soccer server will serve the nets as a stand-in for the world. The action representation is controlled by Petri nets, extensions of the Finite State Machine that add the capability to inherently represent concurrent events.

Information is exchanged between the soccer server and the Petri-Bayes model using User Datagram Protocol (UDP). The datagrams contain the information in s-expression format, which is parsed and interpreted by Atan classes. These classes then update the observation places in the model, run the model, and send action commands back to the simulator when the Petri nets in the model trigger those actions. However, the latest official Atan code release is a bit outdated, and was last tested with RoboCup version 8. This work will use the last version of RoboCup, version 12, and so even though the general format and commands are almost identical between both versions, the Atan code still required an update.

The simulation software serves to ground the action/inference model to a real-world environment, which is necessary for testing the practicality of the network methodology. The Petri-Bayes net editor brings a new style of behavioral design to the RoboCup community, which allows people to construct complex action models in the graphical realm without necessarily having any coding experience.

2. RELATED WORK

Several attempts have been made to deal with real-world action such as planning, reinforcement learning, and Bayes nets, but none of them address all the issues [1].

Research groups all over the world use the RoboCup soccer simulator as a testbed for new methods in real-time action design. A research group at the University of Essex designed a team using reinforcement learning [6]. The world-champion FC Portugal simulation team AI consists primarily of situation based strategic positioning and dynamic role exchange, and tries to maintain a global view of the environment's state by sharing information between agents on the same team [8]. The Royal Melbourne Knights team AI revolves around a system of dynamic roles and responsibilities which reassigns roles between players in order to maximize the number of roles satisfied [3]. A team implemented by a research group at the University of Maryland uses genetic programming to automatically generate functions and algorithms through natural selection [7].

One of the most obvious advantages of so many researchers using the same software to test their approach rests on the fact that most teams have open source versions of their code available for download. This means that modeling techniques can be tested against each other by simply setting them as opponents in the simulator. Techniques can also be easily extended thanks to the open source nature of the community.

3. Background Information

3.1 Bayesian Networks

A Bayesian network is a graphical model in which the nodes represent random variables and the connections between them represent conditional dependence, or more strictly speaking, absence of edges implies conditional independence. In our model, Bayesian places will be represented by double concentric ovals, and Bayesian arcs will be represented by double-headed arrows. Fig. 1 depicts the typical wet grass example. In this case the "wet grass" node is dependent on the rain and sprinklers nodes. From the observed probabilities, we can gather that rain is less probable than sprinkler activity. The conditional probabilities contain much more information, such as "if the sprinklers are off, and it is not raining, the grass is dry with probability x", or "sprinklers are x times more likely to get the grass wet than rain". The advantage of this model is that these beliefs can be numerically represented by probability distributions, which can be updated whenever new information about the environment is received.

Variable elimination algorithms are used to perform inferences on Bayesian networks. These algorithms eliminate (by integration or summation) the non-observed non-query variables one by one by distributing the sum over the product. One of the inherent problems of elimination algorithms is that they have complexity that is exponential in the network's treewidth, so not all the conditional dependencies that exist between variable can always be represented in a Bayesian network [4]. Eq. 1 shows how the product of individual conditional probabilities yields a joint probability distribution of the random variables. Eq. 2

contains the Bayes theorem, which allows for backward inference to take place. In the particular case of E being a node and F being its parents, Eq. 2 expresses the conditional probability of the child given its parents as a function of the probability of the parents given its child (additionally we need the marginal, i.e. unconditional probabilities for both), which is exactly the kind of transformation needed for the backward inference [4].

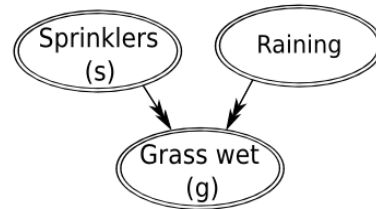


Figure 1: Example of Bayesian Network

$$p(X_1, \dots, X_n) = \prod_{i=1}^n p(X_i | \text{parents}(X_i))$$

Equation 1: Joint Probability Distribution

$$p(E|F) = \frac{p(F|E)p(E)}{p(F)}$$

Equation 2: Bayes theorem

3.2 Petri Networks

A Petri net is a directed bipartite graph with two types of nodes: places and transitions. Places (depicted as circles) can only be connected to transitions (rectangles), and transitions can only be connected to places. A transition "fires" only when all the places that are connected as inputs to it contain at least the number of tokens required for the transition to fire. This numerical threshold is represented by a weight associated with each connection. Upon firing, the transition "consumes" from each input place as many tokens as the input connection weights require, and produces at each output place as many tokens as the output weights specify. Figs. 2a and 2b depict a simple Petri net before and after firing.

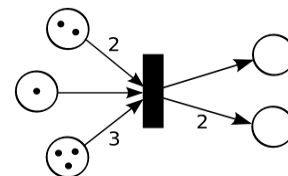


Figure 2a: Petri net with transition ready to fire

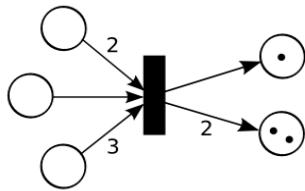


Figure 2b: Same Petri net after transition firing

The Petri nets presented so far are purely discrete, but continuous places and transitions are also an important part of Petri net modeling. In the case of continuous transitions, a firing rate is associated with each transition. This rate is a function of the values contained in the input places, and changes dynamically. Petri nets also use inhibitory and test arcs. Inhibitory arcs activate the transition they connect to only when their input place contains *at most* the value specified by the arc weight; the opposite of normal transitions. Test arcs act as normal arcs, but do not consume tokens from their input place upon transition firings [4].

3.3 Bayes net software

The Bayes net software used for the Petri-Bayes net editor was Bayesian Network tools in Java (BNJ), developed by the Kansas State University Laboratory for Knowledge and Discovery in Databases, and distributed under the GNU General Public License (GPL). BNJ is an open source suite of software tools for research and development using graphical probability models implemented purely in Java. Fig. 3 contains a screenshot of the BNJ graphical user interface.

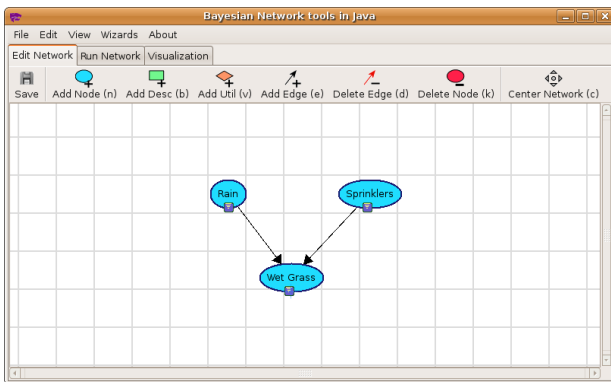


Figure 3: BNJ GUI screenshot

3.4 Petri net software

The Petri net software used in the Petri-Bayes net editor was developed in Java by researchers at the International Computer Science Institute (ICSI) at UC Berkeley. The software facilitates the implementation of Petri nets, providing a toolbox that contains all the Petri variants discussed in section 3.2. The Petri net graphical user interface is the main command center of the Petri-Bayes net editor, from which simulated soccer players

can be connected to the RoboCup soccer server via Atan. Fig. 4 shows a screenshot of the Petri net editor.

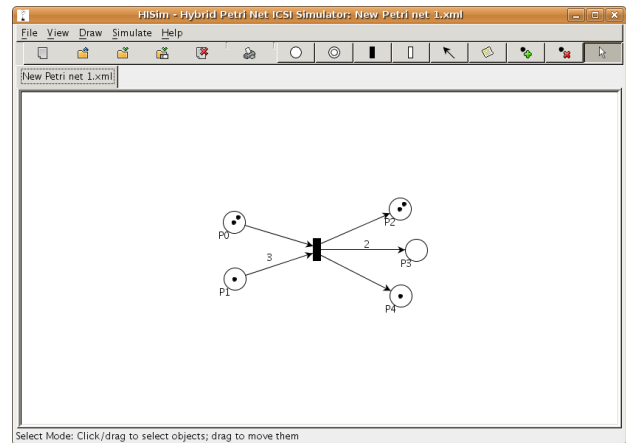


Figure 4: Petri net editor screenshot

3.5 Atan software

Atan is a set of Java classes written by Wolfgang Wagner that facilitate the implementation of AI agents for robocup simulation soccer. Atan handles three main aspects related to soccer server communication: establishing a UDP connection, parsing information from the server (sent in s-expression format), and sending commands to the server (also in s-expression format) [9].

The Atan parser makes use of JavaCC (Java Compiler Compiler), an open source parser generator for the Java programming language. Due to changes in the parameters of certain commands across RoboCup versions, some of the Atan JavaCC code was edited so that it generated a parser compatible with the latest version of RoboCup version 12.

3.6 RoboCup Software

The RoboCup software chosen as a testbed for the Petri-Bayes net model is an open source simulator used in official World Cup Robot Soccer competitions. It has a general client-server architecture which communicates using User Datagram Protocol (UDP) and is implemented in C++. The soccer server is the main object in the software. A set of monitor objects can be connected to the server (again, via UDP), over which the game can be appreciated graphically. Fig. 5 contains a screenshot of a RoboCup monitor. A set of players can also be connected to the server, specifically two teams of eleven players each. This client-server architecture provides great freedom when implementing the AI of the agent, because it can be coded in any programming language that has UDP capabilities. UDP allows data flow to be fast at the expense of the arrival guarantees provided by other protocols such as TCP, but is still regularly used in situations where there is a large real-time data flow for the speed advantage. This constant heavy flow helps ensure that if a UDP packet is lost, the system will not suffer greatly from it, since the next

packet is on its way. (A player is not going to stop working in the simulator because a single datagram was lost during transmission). All the rules, command structure, and code relating to RoboCup are detailed in the RoboCup Manual [2].



Figure 5: RoboCup monitor screenshot

4. Design

The design aspect revolved around two main requirements: getting observations from the simulator to the Petri-Net model, and triggering RoboCup action commands with the model. In order to satisfy the first requirement, Observation Sets (fig. 6) were implemented.

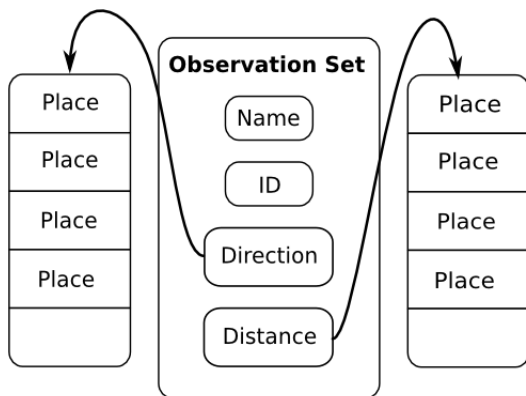


Figure 6: Observation Set

Observation sets embody a specific observation and link the network places that contain those observations to the simulator. Observation Sets consist of a Name, ID, and two lists: one of places that contain the *distance* parameter of that observation, and one of places that contain the *direction* parameter of that observation. The Name refers to a group of observable objects, and the ID specifies which object in that group is being observed by the set. These groups of observable objects identifiable by Name are pictured in fig. 1 of Appendix

A, clustered by yellow rectangles. The individual objects identifiable by ID are represented by the yellow circles in the groups. For example, the Observation Set responsible for the observation of the FLAG_CENTER object in the SeeFlagCenter object group would have "SeeFlagCenter" as its name, FLAG_CENTER as its ID, a list of places containing the direction of the flag relative to the agent's point of view, and a list of places containing the distance to the flag relative to the agent's position. When information regarding the distance and direction of a certain object on the field reaches the Atan module, it finds the corresponding Observation set in a HashMap by using the concatenation of the observed object's Name and ID as a unique key. The Atan module then updates each place in the Observation Set's *distance* and *direction* lists with the new information. Bayes places may be used for triggering Petri transitions by treating them as observed continuous Petri places. Fig. 7 depicts a graphical example of this.

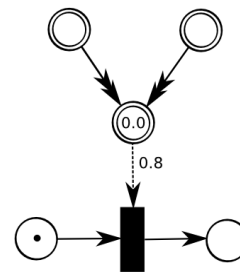


Figure 7: Petri-Bayes interfacing

Once observed information is fed to the Petri-Bayes model, the model is run, and some transitions may fire. The purpose of Parameter Sets is to link Petri net transitions to RoboCup action commands. Each different action has its own parameter set subtype that extends the more general Parameter Set data type. Parameter Sets (fig. 8) are composed of a list of transitions that trigger the associated action (which the main ParameterSet class implements), and a set of parameters unique to each action (which are implemented in the different subclasses).

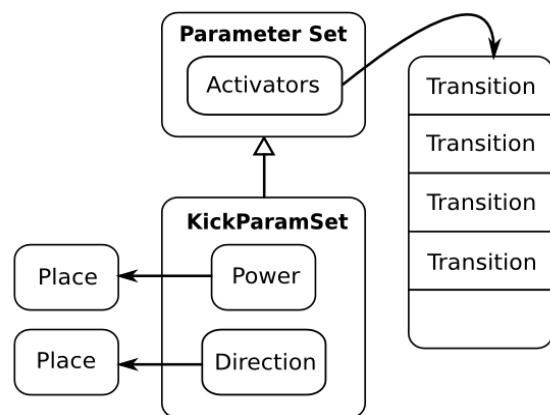


Figure 8: Parameter Set

The “kick” action has two parameters in the simulator: power, and direction. The KickParameterSet object points to two places, one for power and one for direction. The fact that there is a list of activator transitions means that multiple transitions may trigger the a kick action using the same parameter places. When a transition fires in a Petri net, it is added to a list of fired transitions for the current time interval (these lists are cleared at the beginning of each interval). At the end of each time interval, the Atan module retrieves the list of fired transitions from the Petri-Bayes model and sends the corresponding action commands to the RoboCup server. The command's parameters are retrieved from the Parameter set to which the fired transition belongs. If the fired transition does not belong to any parameter set, then no command is executed as a result of its firing. The main execution loop just described is pictured in fig. 9.

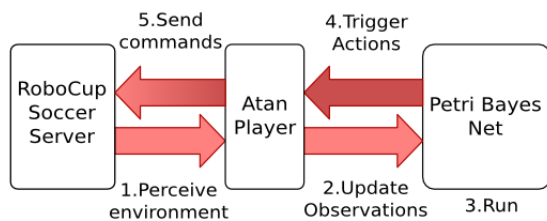


Figure 9: System execution loop

The Petri-Bayes net *run* step can also be described in terms of a cycle. First observed probabilities are updated in the corresponding observation places. Then conditional probabilities are computed in the Bayes net. Petri places are then updated with the newly inferred information, and finally enabled transitions are fired in the Petri net. This cycle is pictured in fig. 10 as an extension of fig. 9.

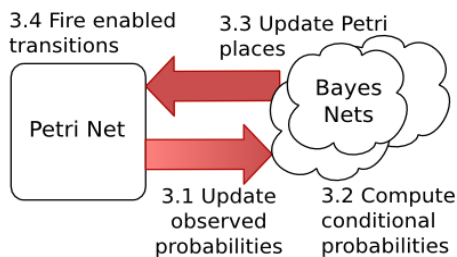


Figure 10: Petri-Bayes net run cycle

6. Discussion

The link between the Petri-Bayes editor and the RoboCup soccer server was successful. Action routines modeled in the editor were faithfully carried out in the simulator. Saving and loading capabilities were added to the parameter and observation sets so that when Petri-Bayes nets are loaded, the transitions and places that belonged to a particular set are still part of that set. In order to test the basic functionality of the system, a simple Petri net was constructed that modeled the

following action sequence: dash forward incrementally fast until the ball is seen, then stop as fast as possible. This Petri net is pictured in Fig. 2 of Appendix A. The result was a simulated soccer player that ran incrementally fast, and when the ball was seen by him, he began to stop, eventually stopping. This example showed that not only do basic action models work in the simulator, but that more complex simulator phenomena such as inertia must be taken into account when designing the models. For the moment, only basic actions have been implemented in the model, but the simulator actions seem to follow the execution routines faithfully.

The graphical editor interface provides an intuitive design toolbox for modeling agent behavior in the RoboCup simulation. Petri net transitions received several new options in their right-click menu. Specifically, transitions can now be added or removed from a Parameter Set as an action trigger, and their corresponding Parameter Set can be seen viewed from their right-click menu. Petri net places received the right-click menu option of adding or removing them from a Parameter Set as the holder of a specific parameter related to the action that the set is responsible for. If a place belongs to a Parameter Set, the set can be seen from the right-click menu. Petri places also received the new right-click menu option of containing observation values, namely distance and direction. This option uses Observation Sets to carry out its function by simply adding the specified place to the corresponding place list in the corresponding set.

7. Conclusion

Petri nets and Bayes nets can now be successfully used to model the real-time behavior of RoboCup soccer agents. Complex soccer strategies have not yet been implemented using the editor, but can now be designed, thanks to the link established between the nets and the simulator. Simple low-level tasks such as running and stopping at the site of a specific observation were successful.

8. References (alphabetical order!)

- [1] L. Barrett. Thesis Proposal: Structured Actions With Structured Bayes Nets and Petri Nets. 2007.
- [2] M. Chen, K. Dorer, E. Foroughi, F. Heintz, Z. Huang, S. Kapetanakis, K. Kostiadis, J. Kummeneje, J. Murray, I. Noda, O. Obst, P. Riley, T. Steffens, Y. Wang, and X. Yin. *Users Manual: RoboCup Soccer Server*. 2003.
- [3] S. Ch'ng and L. Padgham. Team Description: Building Teams Using Roles, Responsibilities, and Strategies. *RoboCup-97: Robot Soccer World Cup I*. 458-466, 1998.
- [4] M. Jordan. *An Introduction to Probabilistic Graphical Models*. University of California, Berkeley. 2003.

- [5] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, E. Osawa, and H. Matsubara. Robocup: A challenge problem for AI. *AI Magazine*, 18(1):73-85, 1997.
- [6] K. Kostiadis and H. Hu. Reinforcement Learning and Cooperation in a Simulate Multi-agent System. *Proceedings of the 1999 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2:990-995, 1999.
- [7] S. Luke, C. Hohn, J. Farris, G. Jackson, and J. Hendler. Co-evolving Soccer Softbot Team Coordination with Genetic Programming. *RoboCup-97: Robot Soccer World Cup I*. 398-411, 1998.
- [8] L.P. Reis, N. Lau. FC Portugal Team Description: RoboCup 2000 Simulation League Champion. *RoboCup 2000: Robot Soccer World Cup IV*. 29-40, 2001.
- [9] W. Wagner. Atan RoboCup Interface. Available online at: <http://atan1.sourceforge.net/index.html>. 2004.

9. Acknowledgements

I would like to thank the members of the International Computer Science Institute (ICSI) at the University of California, Berkeley, for their continuous support of my research. Special thanks go to my faculty mentor, Professor Jerome A. Feldman, who gave me the opportunity to participate in Berkeley's SUPERB-CISE 2008 Program conducting research with his group, to my graduate mentor, Leon Barrett, who helped me throughout every stage of this research project, from design to review, and to the SUPERB staff for helping me establish the necessary connections to carry out this research.

Appendix A

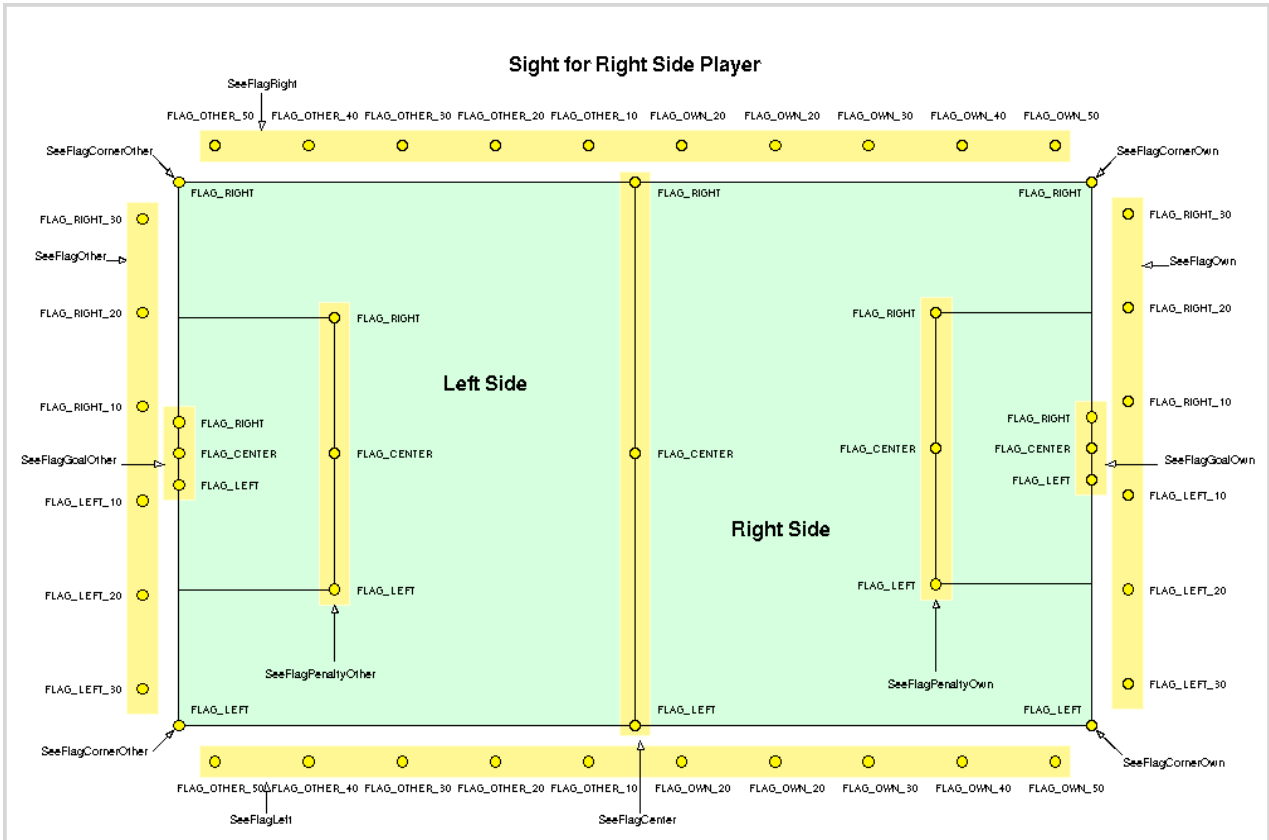


Figure 1: Observable RoboCup field objects

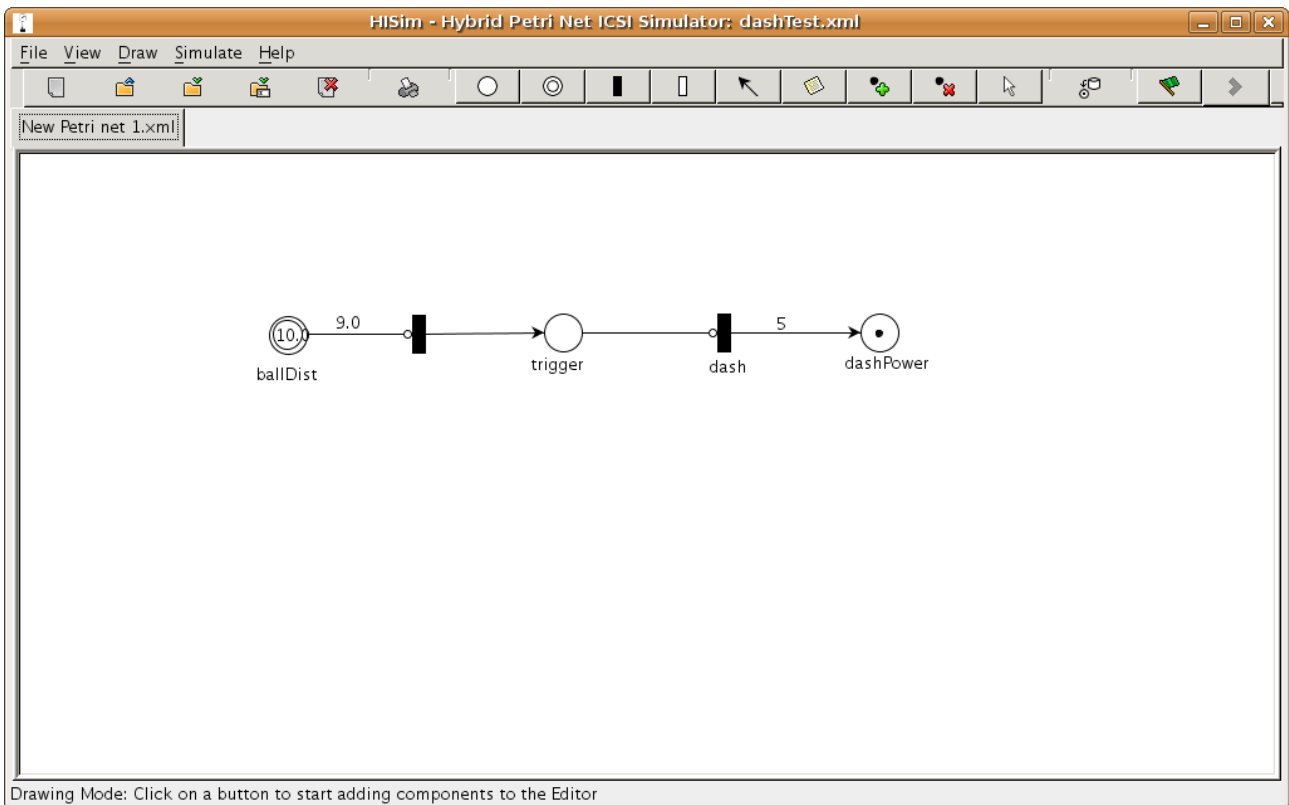


Figure 2: Test Petri net designed to test basic simulator control