Proceedings of the 1998 IEEE/RSJ
Intl. Conference on Intelligent Robots and Systems
Victoria, B.C., Canada • October 1998

A Constraint-based Controller for Soccer-playing Robots

Yu Zhang and Alan K. Mackworth

Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada, V6T 1Z4
E-mail: yzhang,mack@cs.ubc.ca

Abstract

Soccer meets the requirements of the Situated Agent approach and as a task domain is sufficiently rich to support research integrating many branches of robotics and AI. A robot is an integrated system, with a controller embedded in its plant. A robotic system is the coupling of a robot to its environment. Robotic systems are, in general, hybrid dynamic systems, consisting of continuous, discrete and event-driven components. Constraint Nets provide a semantic model for modeling hybrid dynamic systems. Controllers are embedded constraint solvers that solve constraints in real-time. A controller for our new softbot soccer team, UBC Dynamo98, has been modeled in Constraint Nets, and implemented in Java, using the Java Beans architecture. The paper demonstrates that the formal Constraint Net approach is a practical tool for designing and implementing controllers for robots in multi-agent real-time environments.

1 Background and Introduction

The Good Old Fashioned Artificial Intelligence and Robotics (GOFAIR) [3] research paradigm has shaped the area of intelligent robotics since the time of the robot Shakey. Some of the typical fundamental assumptions made about the world were that there is only one agent, that the environment is static unless the agent changes it, that actions are discrete and are carried out sequentially and that the world the robot inhabits can be accurately and exhaustively modeled by the robot. These assumptions proved to be overly restrictive and ultimately sterile. In the usual dynamic of the scientific dialectic, a new movement has emerged as a synthesis of GOFAIR and "Nouvelle AI": the Situated Agent approach. A situated agent is a real physical system grounded and embedded in a real world, here and now, acting and reacting in real-time. Mackworth [3] proposed that playing soccer be a paradigmatic task domain since it breaks with nearly all of the restrictive assumptions on which GOFAIR is based and meets the requirements of the Situated Agent approach. The soccer domain has the following characteristics:

1. Neutral, friendly, and hostile agents
2. Inter-agent cooperation and communication
3. Real-time interaction
4. Dynamic environment
5. Real and partially unpredictable world
6. Objective performance criteria
7. Repeatable experiments

Soccer as a task domain is sufficiently rich to support research integrating many branches of robotics and AI [4].

To satisfy the need for a common environment, the Soccer Server was developed by Noda Itsuki [1] to make it possible to compare various algorithms for multi-agent systems. Because the physical abilities of the players are all identical in the server, individual and team strategies are the focus of comparison. The Soccer Server is used by many researchers and has been chosen as the official simulator for the RoboCup Simulation League [2].

The Constraint Net framework (CN) is a formal model for robotic systems and behaviours [7]. CN provides a theoretical foundation for systems design and analysis. CN captures the most general structure of dynamic systems so that systems with discrete

0-7803-4465-0/98 $10.00 © 1998 IEEE

and continuous time, discrete and continuous variables, and asynchronous as well as synchronous event structures can be modeled in a unitary framework [5]. A controller for our new softbot soccer team, *UBC Dynamo98*, has been developed using CN.

The rest of the paper describes CN and how we use it to model and build the controller for our soccer-playing softbot *UBC Dynamo98*. Section 2 introduces Constraint Nets and the CN architecture of robotic systems. Section 3 introduces the CN model of the controller for our soccer-playing softbot. Section 4 discusses constraint-based control and shows how the controller satisfies the constraints in the soccer domain. Section 5 concludes the paper.

2 Modeling in Constraint Nets

A robot is an integrated system, with a controller embedded in its plant. A robot controller (or control system) is a subsystem of a robot, designed to regulate its behavior to meet certain requirements. A robotic system is the coupling of a robot to its environment. Robotic systems are, in general, hybrid dynamic systems, consisting of continuous, discrete and event-driven components. The dynamic relationship of a robot and its environment is called the behavior of the robotic system.

Constraint Nets (CN), a semantic model for hybrid dynamic systems, can be used to develop a robotic system, analyze its behavior and understand its underlying physics. Using this model, we can characterize the components of a system and derive the behavior of the overall system. CN is an abstraction and generalization of dataflow networks. Any (causal) system with discrete/continuous time, discrete/continuous (state) variables, and asynchronous/synchronous event structures can be modeled. Furthermore, a system can be modeled hierarchically using aggregation operators; the dynamics of the environment as well as the dynamics of the plant and the controller can be modeled individually and then integrated [5].

A constraint net consists of a finite set of locations, a finite set of transductions and a finite set of connections. Formally, a *constraint net* is a triple $CN = \langle Lc, Td, Cn \rangle$, where Lc is a finite set of *locations*, Td is a finite set of labels of *transductions*, each with an *output port* and a set of *input ports*, Cn is a set of *connections* between locations. A location can be regarded as a wire, a channel, a variable, or a memory cell. Each transduction is a causal mapping from inputs to outputs over time, operating according to a certain reference time or activated by external events.

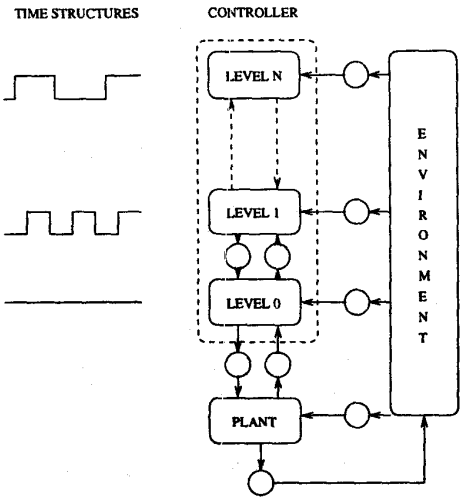

Figure 1: Abstraction hierarchy

Semantically, a constraint net represents a set of equations, with locations as variables and transductions as functions. The *semantics* of the constraint net, with each location denoting a trace, is the least solution of the set of equations. For *trace* and some other basic concepts of dynamic systems, the reader is referred to [8].

Given CN, a constraint net model of a dynamic system, the abstract behavior of the system is the semantics of CN, denoted $[\![CN]\!]$, i.e., the set of input/output traces satisfying the model.

A complex system is generally composed of multiple components. A *module* is a constraint net with a set of locations as its interface. A constraint net can be composed hierarchically using modular and aggregation operators on modules. The semantics of a system can be obtained hierarchically from the semantics of its subsystems and their connections.

A constraint net is depicted by a bipartite graph where locations are depicted by circles, transductions by boxes and connections by arcs. A module is depicted by a box with rounded corners.

A control system is modeled as a module that may be further decomposed into a hierarchy of interactive modules (Fig. 1). The higher levels are typically composed of event-driven transductions and the lower levels are typically analog control components. The bottom level sends control signals to various effectors, and at the same time, senses the state of effectors. Control signals flow down and the sensing signals flow

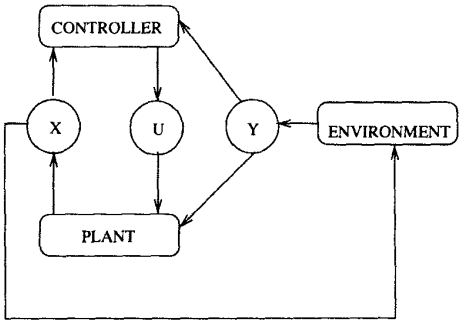

Figure 2: A robotic system

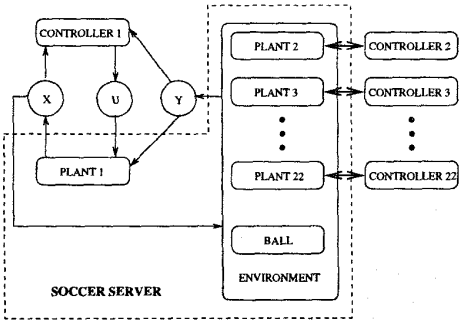

Figure 3: The soccer-playing softbot system

up. Sensing signals from the environment are distributed over levels. Each level is a black box that represents the causal relationship between the inputs and the outputs. The inputs consist of the control signals from the higher level, the sensing signals from the environment and the current states from the lower level. The outputs consist of the control signals to the lower level and the current states to the higher level. Usually, the bottom level is implemented by analog circuits that function with continuous dynamics and the higher levels are realized by distributed computing networks.

Furthermore, the environment of the robot can be modeled as a module as well. A robotic system can be modeled as an integration of a plant, a controller and an environment (Fig. 2). A plant is a set of entities which must be controlled to achieve certain requirements, for example, a car with throttle and steering. A controller is a set of sensors and actuators, which, together with software/hardware computational systems, (partially) senses the states of the plant (X) and the environment (Y), and computes the desired control inputs (U) to actuate the plant. An environment is a set of entities beyond the (direct) control of the controller, with which the plant may interact. For example, obstacles to be avoided and objects to be reached can be considered as the *environment* of a robotic system.

In most cases, desired goals, safety requirements and physical restrictions of a robotic system can be specified by a set of constraints on variables $U \cup X \cup Y$. The controller is then synthesized to regulate the system to satisfy the set of constraints. The semantics (or behavior) of the system is the solution of the following equations:

$$X = PLANT(U, Y), \quad (1)$$

$$U = CONTROLLER(X, Y), \quad (2)$$

$$Y = ENVIRONMENT(X). \quad (3)$$

Note that $PLANT$, $CONTROLLER$ and $ENVIRONMENT$ are transductions mapping input traces to output traces (not simple functions), and the solution gives X, Y and U as tuples of traces (not values).

3 The CN Architecture of the Controller for a Soccer-playing Softbot

The soccer-playing softbot system is modeled as an integration of the soccer server and the controller (Fig. 3). The soccer server provides 22 soccer-playing softbots' plants and the ball. Each softbot can be controlled by setting its throttle and steering. When the softbot is near the ball (within 2 meters), it can use the kick command to control the ball's movement. For the controller for one of the soccer-playing softbots, the rest of the players on the field and the ball are considered as its environment. The sensor of the controller determines the state of the plant (position and direction) by inference from a set of landmarks it 'sees'. The rest of the controller computes the desired control inputs (throttle and steering) and sends them to the soccer server to actuate the plant to move around on the field or kick or dribble the ball.

For the soccer-playing softbot, we have designed a three-level controller. The lowest level is the Effector&Sensor. It receives ASCII sensor information from the soccer server then translates it into the World model. It also passes commands from the upper level down to the soccer server. The middle level is the Executor. It tries to translate the action (goal) which

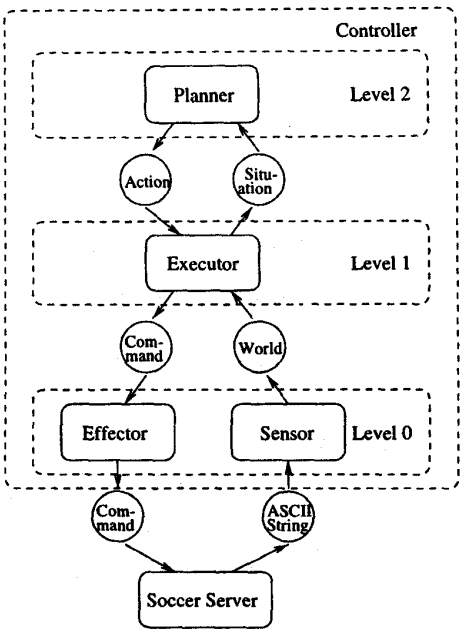

Figure 4: The soccer-playing controller hierarchy

comes from the upper level into a sequence of commands and sends them to the lowest level. The Executor also evaluates the situation and sends it to the top layer (Planner). The highest level is the Planner. It decides which action (goal) to take based on the current situation and it may also consider the next action assuming the current action will be correctly finished on schedule.

The controller is composed of four CN modules. The Effector module combines with the Sensor module to form the lowest level Effector&Sensor. The Executor module forms the middle level and the Planner module forms the highest level (Fig. 4).

The CN controller is written in Java because it is object-oriented, provides features like easy thread programming and an effective event mechanism. The Java Beans component architecture is used here to implement the CN modules.

Events are one of the core features of the Java Beans architecture. Conceptually, events are a mechanism for propagating state notifications between a *source* object and one or more target *listener* objects. Under the new AWT event model, an event listener object can be registered with an event source. When the event source detects that something interesting has happened it calls an appropriate method in the event listener object.

CN model is a data-flow model; each CN module can be run concurrently on different processors to improve the speed of the controller. Since these modules are event-driven and fixed-sample-time-driven, they are best implemented as Java threads to improve efficiency on a single CPU too. If no event arrives, they go to sleep so the CPU can deal with other softbots. In such a multi-threaded environment where several different threads may be simultaneously delivering events and/or calling methods and/or processing event objects and/or setting properties, special considerations are needed to make sure these beans properly coordinate their behaviour, using wait/notify and synchronization mechanisms.

The Sensor module wakes up when new information arrives. It then processes the ASCII information from soccer server, updates the world model, and sends an event to the Executor. The Sensor goes to sleep when there is no information waiting on its socket.

The Executor module receives the event from the Sensor, then it processes the world model and updates situation states. These situation states tell the Planner if it can kick the ball, if the ball is in its sight, if it is the nearest player to the ball, if there are obstacles on its way, the action from the Planner has finished or not, and so on. Any change of situation creates an event and triggers the higher level Planner module. This part of the Executor runs in the same thread as the Sensor module.

The main part of the Executor executes actions passed down from the Planner. It wakes up when it receives an action event from the Planner module. It produces a sequence of commands which are supposed to achieve goals (actions) when they are performed. Commands such as *kick*, *dash*, and *turn* are sent to the Effector's *Movement_command* buffer. Other commands are sent to the Effector's *Sensing_command* buffers: the *Say_message* buffer, the *Change_view* buffer, and the *Sense_body* buffer. The Executor goes to sleep when there is no action waiting for its processing.

The Planner module wakes up when triggered by a situation-changed event from the Executor. It then produces actions and pushes them into Executor's action buffer and sends an event to trigger the Executor to execute actions. Then it goes to sleep until a new event arrives.

The Effector module is a fixed-sample-time-driven module. Every 100ms, it gets one command from each non-empty buffer and sends them to the soccer server.

This is a hybrid control system because it has both

event-driven and fixed-sample-time-driven modules.

4 Constraint-Based Control for Soccer-playing Softbot

Constraints are considered to be relations on a set of state variables; the solution set of the constraints consists of the state variable tuples that satisfy all the constraints. The behavior of a dynamic system is constraint-based if the system is asymptotically stable at the solution set of the given constraints, i.e., whenever the system diverges because of some disturbance, it will eventually return to the set satisfying the constraints. Most robotic systems are constraint-based, where the constraints may include physical limitations, environmental restrictions, and safety and goal requirements. Most learning and adaptive dynamic systems exhibit some forms of constraint-based behaviors as well [6].

A controller is an *embedded constraint solver* if the controller, together with the plant and the environment, satisfies the given constraint-based specification.

In the framework for control synthesis, constraints are specified at different levels on different domains, with the higher levels more abstract and the lower levels more plant-dependent. A control system can also be synthesized as a hierarchy of interactive embedded constraint solvers. Each abstraction level solves constraints on its state space and produces the input to the lower level. Typically the higher levels are composed of digital/symbolic event-driven control derived from discrete constraint methods and the lower levels embody analog control based on continuous constraint methods [5].

The Executor module can be seen as an embedded constraint solver on its world state space. It solves the constraint-based requirements passed down from the higher layer Planner module. For example, if the action from the Planner is to go to (x_d, y_d) and the position state variables of the robot soccer player are (x, y), the set of constraints are $x = x_d, y = y_d$. If the action from the Planner is to intercept the ball at (x_b, y_b, vx_b, vy_b), and the state variables of the robot soccer player are (x_p, y_p, vx_p, vy_p), the set of constraints are $x_p + vx_p * t = x_b + vx_b * t$ and $y_p + vy_p * t = y_b + vy_b * t$.

The Planner module can be seen as an embedded constraint solver on its situation state space. The ultimate constraint here is: the number of goals scored should be more than its opponent's. To satisfy this ultimate constraint, the robot has to satisfy a series of other constraints first.

These constraints have their priorities. The constraints with higher priority must be solved earlier. The constraint of knowing its position and the ball's should be solved first. Then the robot will try to solve the constraints of collision and offside. In order to win, the robot will consider some other constraints, such as, its own team's time in possession of the ball should be greater than its opponent's team, the ball should be near enough to the opponent's goal, the ball should be as far away as possible from its own goal, and the ball should be kicked into opponent's goal instead of its own goal.

It chooses actions to satisfy the constraints at this level. When robot loses its own position or the ball's position for a certain amount of time, it sends *find_me* or *find_ball* actions down to the Executor. When the robot senses that it will collide with other players, it sends *avoid_collision* action down to the Executor. It also sends down *avoid_offside* down to the Executor if it finds itself is at offside position. The robot tries to *intercept* the ball if it senses that it is nearer to the ball than its teammates, if not, it goes to a suitable position to *assist* its teammate's interception.

If the robot gets the ball, it has to choose where to kick. The best action should optimally satisfy the constraints above. Here we have two problems. First, the robot can't be certain that an action will satisfy a constraint because the soccer server provides a noisy, dynamic world. For example, it's impossible for the robot to choose a kick direction which makes sure that its teammates will get the ball first. We can only say that if the robot chooses to kick in this direction, the probability of teammates getting the ball first is high. Second, the robot can't find a kick direction that can maximize all the probabilities of satisfying all the constraints. For example, if the robot chooses the kick direction which makes the probability of its teammates getting the ball very high, the ball might be kicked away from its opponent's goal and near its own goal.

We solve this by setting weights for these constraints and combine these constraints into a *utility function* , which assigns a single number to express the desirability of an action. The Planner chooses the action with the highest utility.

$$U(a) = \sum_i k_i * P_i(a)$$

$U(a)$ is the action a's utility. $P_i(a)$ is the probability of satisfying the constraint i when taking the action a. k_i is the weight for the constraint i.

These weights can be set by hand. They can also be tuned by learning methods, such as reinforcement learning. We have designed a coach program using an evolutionary algorithm to adjust these weights and other parameters in the controller.

Also the utility function $U(a)$ need not be *linear*, it might be obtained by using neural networks learning.

A series of soccer-playing experiments were performed for evaluating the constraint-based controller. In one experiment three different versions of the controller were compared to find out which constraint is more important. The controller for team 1 considers all the constraints. The controller for team 2 only considers the constraint of kicking the ball into the opponent's goal. The controller for team 3 only considers the constraint of letting its teammates get the ball first. Table 1 shows the scores for all the matches.

Table 1: Scores for soccer games (Row:Column)

vs	Team 1	Team 2	Team 3
Team 1	-	2:1	6:1
Team 2	1:2	-	5:0
Team 3	1:6	0:5	-

From this experiment, we learn that the constraint of shooting at goal is more important than the constraint of passing the ball to its teammates and the more constraints are considered, the better the performance.

5 Summary and Conclusions

Constraint Nets (CN), a semantic model for hybrid dynamic systems, can be used to develop a robotic system, analyze its behavior and understand its underlying physics.

The soccer-playing softbot system is modeled as an integration of the soccer server and the controller. The three-level controller is composed of four modules. The Effector module combines with the Sensor module to form the lowest level Effector&Sensor. The Executor module forms the middle level and the Planner module forms the highest level. The controller is written in Java. The Java Beans component architecture is used here to implement the CN modules and we use the Java event mechanism to implement communication among these CN modules. They are implemented in Java threads to improve efficiency.

The controller for soccer-playing softbot is synthesized as a hierarchy of interactive embedded constraint solvers. Each level solves constraints on its state space and produces the input to the lower level.

In short, we have demonstrated that the CN model is a formal and practical tool for designing and implementing, in Java, constraint-based controllers for robots in multi-agent, real-time environments.

Acknowledgments

We wish to thank Ying Zhang for valuable discussions and suggestions.

References

[1] Noda Itsuki. Soccer Server System. Available at http: //ci.etl.go.jp/ noda/soccer/server.html.

[2] Hiroaki Kitano. Robocup. Available at http: //www.robocup.org/RoboCup/New/index.html.

[3] A. K. Mackworth. On seeing robots. In A. Basu and X. Li, editors, *Computer Vision: Systems, Theory, and Applications*, pages 1–13. World Scientific Press, Singapore, 1993.

[4] M. Sahota and A. K. Mackworth. Can situated robots play soccer? In *Proc. Artificial Intelligence 94*, pages 249 – 254, Banff, Alberta, May 1994.

[5] Ying Zhang and A. K. Mackworth. Synthesis of hybrid constraint-based controllers. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 552 – 567. Springer Verlag, 1995.

[6] Ying Zhang and A. K. Mackworth. Constraint Programming in Constraint Nets. Principles and Practice of Constraint Programming, MIT Press, 1995, p.49–68.

[7] Ying Zhang and A. K. Mackworth. Constraint Nets: A Semantic Model for Hybrid Dynamic Systems. Journal of Theoretical Computer Science, Vol. 138, No. 1, 1995, p.211–239, Special Issue on Hybrid Systems.

[8] Ying Zhang. A foundation for the design and analysis of robotic systems and behaviors. Technical Report 94-26, Department of Computer Science, University of British Columbia, 1994. Ph.D. thesis.