The Dynamics of Intelligence: A Constraint-Based Architecture for Situated Visual Agents

Alan K. Mackworth and Yu Zhang

Laboratory for Computational Intelligence,
Department of Computer Science,
University of British Columbia,
Vancouver, B.C. V6T 1Z4, Canada
E-mail: mack,yzhang@cs.ubc.ca
WWW: www.cs.ubc.ca/spider/mack,yzhang

Abstract Our goal is to develop practical and formal design methodologies for building integrated perceptual agents. The methodologies are evolving dialectically. The symbolic methods of Good Old-Fashioned Artificial Intelligence and Robotics (GOFAIR) constitute the original thesis. The antithesis is reactive Insect AI. The emerging synthesis, Situated Agents, needs formal rigor and practical tools. A robot is a hybrid intelligent dynamical system, consisting of a controller coupled to its plant. The Constraint Net (CN) model of Zhang and Mackworth is a formal and practical model for building hybrid intelligent systems as Situated Agents. Even though a robotic system is, generally, a hybrid system, its CN model is unitary. We advocate an architecture for robot controllers consisting of multi-layer constraint-satisfying modules. We have developed a testbed for multiple, visually-controlled, cheap robot vehicles performing a variety of tasks, including playing soccer. Soccer meets the scientific requirements of the Situated Agent approach and as a task domain is sufficiently rich to support research integrating many branches of robotics and AI. The controllers for our new softbot soccer team, *UBC Dynamo98*, are modeled in CN and implemented in Java, using the Java Beans architecture. We demonstrate that the formal Constraint Net approach is a practical tool for designing and implementing controllers for perceptual robots in multi-agent real-time environments.

1 Introduction

We need practical and formal design methodologies for building integrated perceptual agents. Here we argue for a formal approach to the emerging synthesis, Situated Agents. The approach is based on the view that any agent or robot is a hybrid intelligent dynamical system, consisting of a controller coupled to its plant. The robot is symmetrically coupled to a dynamic environment, forming a robotic system.

Similarly, knowledge-based image interpretation needs to be re-interpreted. The traditional Good Old-Fashioned Artificial Intelligence and Robotics (GOFAIR) approach proposes that domain-specific knowledge is used by the robot/agent at run-time to disambiguate the retinal array into a rich world representation. The argument is that the impoverishment and ambiguity of the visual stimulus array must be supplemented by additional knowledge. This approach has failed to make substantial progress for several

reasons. One difficulty is the engineering problem of building robots by integrating off-line knowledge-based vision systems with on-line control-based motor systems. Especially in active vision systems [2] this integration is difficult, ugly and inefficient [8]. Because of such objections, some in the AI-robotics community have rejected the knowledge-based approach adopting instead an *ad hoc* Gibsonian situated approach to perception that exploits regularities of the particular environmental niche of the robot [6, 1, 5, 4]. In [9], Mackworth argued that, with a radical re-interpretation of 'knowledge-based', we *can* design, build and verify quick and clean knowledge-based situated robot vision systems.

2 The Design Problem

The robot design problem is formidable, regardless of whether the robot is designed or modified by a human, by nature (evolution), by another robot (bootstrapping), or by itself (learning). A robot is, typically, a hybrid intelligent system, consisting of a controller coupled to its plant. The controller and the plant each consist of discrete-time, continuous-time or event-driven components operating over discrete or continuous domains. The controller has perceptual subsystems that can (partially) observe the state of the environment and the state of the plant.

Robot design methodologies are evolving dialectically [8]. The symbolic methods of GOFAIR constitute the original thesis. The antithesis is reactive Insect AI. The emerging synthesis, Situated Agents, has promising characteristics, but needs formal rigor and practical tools. The critiques and rejection, by some, of the GOFAIR paradigm have given rise to the Situated Agent approaches of Rosenschein and Kaebling [10], Brooks [4], Ballard [2], Winograd and Flores [12], Lavignon and Shoham [7], Zhang and Mackworth [14, 8] and many others.

3 The Robot Soccer Challenge

Theory is vacuous without an appropriate application to drive designs, experiments and implementations. In 1992, Mackworth proposed robot soccer as a grand challenge problem for the field [8] since it has the task characteristics that force us to deal all these issues in a practical way for a percepetual, collaborative, real-time task with clear performance criteria. At the same time, Mackworth also described the first implemented system for playing robot soccer. Since then it has been a very productive environment both for our laboratory [3, 11, 9, 17, 19, 18] and for many other groups around the world through the influential RoboCup initiative [21], stimulating research toward the goal of building vision and robotic systems that are uniform, clean, powerful and practical.

4 Constraint Nets

The Constraint Net (CN) model [15] is a formal and practical model for building hybrid intelligent systems as Situated Agents. In CN, a robotic system is modelled formally as a symmetrical coupling of a robot with its environment. Even though a robotic system

is, typically, a hybrid dynamic system, its CN model is unitary. Most other robot design methodologies use hybrid models of hybrid systems, awkwardly combining off-line computational models of high-level perception, reasoning and planning with on-line models of low-level sensing and control.

CN is a model for robotic systems software implemented as modules with I/O ports. A module performs a transduction from its input traces to its output traces, subject to the principle of causality: an output value at any time can depend only on the input values before, or at, that time. The model has a formal semantics based on the least fixpoint of sets of equations [15]. In applying it to a robot operating in a given environment, one separately specifies the behaviour of the robot plant, the robot control program, and the environment. The total system can then be shown to have various properties, such as safety and liveness, based on provable properties of its subsystems. This approach allows one to specify and verify models of embedded control systems. Our goal is to develop it as a practical tool for building real, complex, sensor-based robots. It can be seen as a development of Brooks' subsumption architecture [4] that enhances its modular advantages while avoiding the limitations of the augmented finite state machine approach.

A robot situated in an environment is modeled as three machines: the robot plant, the robot control and the environment. Each is modeled separately as a dynamical system by specifying a CN with identified input and output ports. The robot is modeled as a CN consisting of a coupling of its plant CN and its control CN by identifying corresponding input and output ports. Similarly the robot CN is coupled to the environment CN to form a closed robot-enviroment CN.

Although CN can carry out traditional symbolic computation on-line, such as solving Constraint Satisfaction Problems and path planning, notice that much of the symbolic reasoning and theorem-proving may be outside the agent, in the mind of the designer. GOFAIR does not make this distinction, assuming that such symbolic reasoning occurs explicitly in, and only in, the mind of the agent.

The question "Will the robot do the right thing?" [14] is answered if we can:

1. model the coupled robotic system at a suitable level of abstraction,
2. specify the required global properties of the system's evolution, and
3. verify that the model satisfies the specification.

In CN the modelling language and the specification language are totally distinct since they have very different requirements. The modelling language is a generalized dynamical system language. Two versions of the specification language, Timed Linear Temporal Logic [17] and Timed $\forall$-automata [13], have been developed with appropriate theorem-proving and model-checking techniques for verifying systems.

5 Constraint-Satisfying Controllers

Many robots can be designed as on-line constraint-satisfying devices [13, 16, 17]. A robot in this restricted scheme can be verified more easily. Moreover, given a constraint-based specification and a model of the plant and the environment, automatic synthesis of a

correct constraint-satisfying controller becomes feasible, as shown for a simple ball-chasing robot in [17].

A constraint is simply a relation on the phase space of the robotic system, which is the product of the controller, plant and environment spaces. A controller is defined to be *constraint-satisfying* if it, repeatedly, eventually drives the system into an ϵ-neighborhood of the constraint using a constraint satisfaction method such as gradient descent.

A constraint-satisying controller may be *hierarchical* with several layers of controller above the plant. In this case each layer must satisfy the constraints appropriate to the layer, defined on its state variables. The layers below each layer present to that layer as a virtual robot plant in a suitably abstract state space [17, 18].

6 Soccer-Playing Robots

The ideas presented here have been developed and tested by application to the challenge of designing, building and verifying active perception systems for robot soccer players with both off-board and on-board vision systems.

In the Dynamo (Dynamics and Mobile Robots) Project in our laboratory, we have experimented, since 1991, with multiple mobile robots under visual control. The Dynamite testbed consists of a fleet of radio-controlled vehicles that receive commands from a remote computer. Using our custom hardware and a distributed MIMD environment, vision programs are able to monitor the position and orientation of each robot at 60 Hz; planning and control programs generate and send motor commands at the same rate. This approach allows umbilical-free behaviour and very rapid, lightweight fully autonomous robots. Using this testbed we have demonstrated various robot tasks [3], including playing soccer [11].

One of the Dynamo robots, Spinoza, is a self-contained robot consisting of an RWI base with an RGB camera on a pan-tilt platform mounted on top and binocular monochrome stereo cameras in the body. As a illustration of these ideas consider the task for Spinoza of finding, tracking and chasing a soccer ball, using the pan-tilt camera. After locating the moving ball Spinoza is required to move to within striking distance of the ball and maintain that distance. The available motor commands control the orientation of the base, the forward movement of the base, and the pan and tilt angles of the camera. The parameters can be controlled in various relative/absolute position modes or rate mode. The available rate of pan substantially exceeds the rate of body rotation. A hierarchical constraint-based active-vision controller can be specified for Spinoza that will achieve and maintain the desired goal subject to safety conditions such as staying inside the soccer field, avoiding obstacles and not accelerating too quickly. If the dynamics of Spinoza and the ball are adequately modelled by the designer then this constraint-based vision system will be guaranteed to achieve its specification.

We have recently extended these ideas to build multi-layer constraint-satisfying controllers for a complete soccer team [19]. The controllers for our new softbot soccer team, *UBC Dynamo98*, are modeled in CN and implemented in Java, using the Java Beans architecture [18]. They control the soccer players' bodies in the Soccer Server developed by Noda Itsuki[20] for the RoboCup Simulation League [21]. This experiment provides

evidence that CN is a uniform, clean, powerful and practical design framework for perceptual robots. In the rest of this paper we describe in more detail the constraint-based architecture and the *UBC Dynamo98* system built in it.

7 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 [17].

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.

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 [22].

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

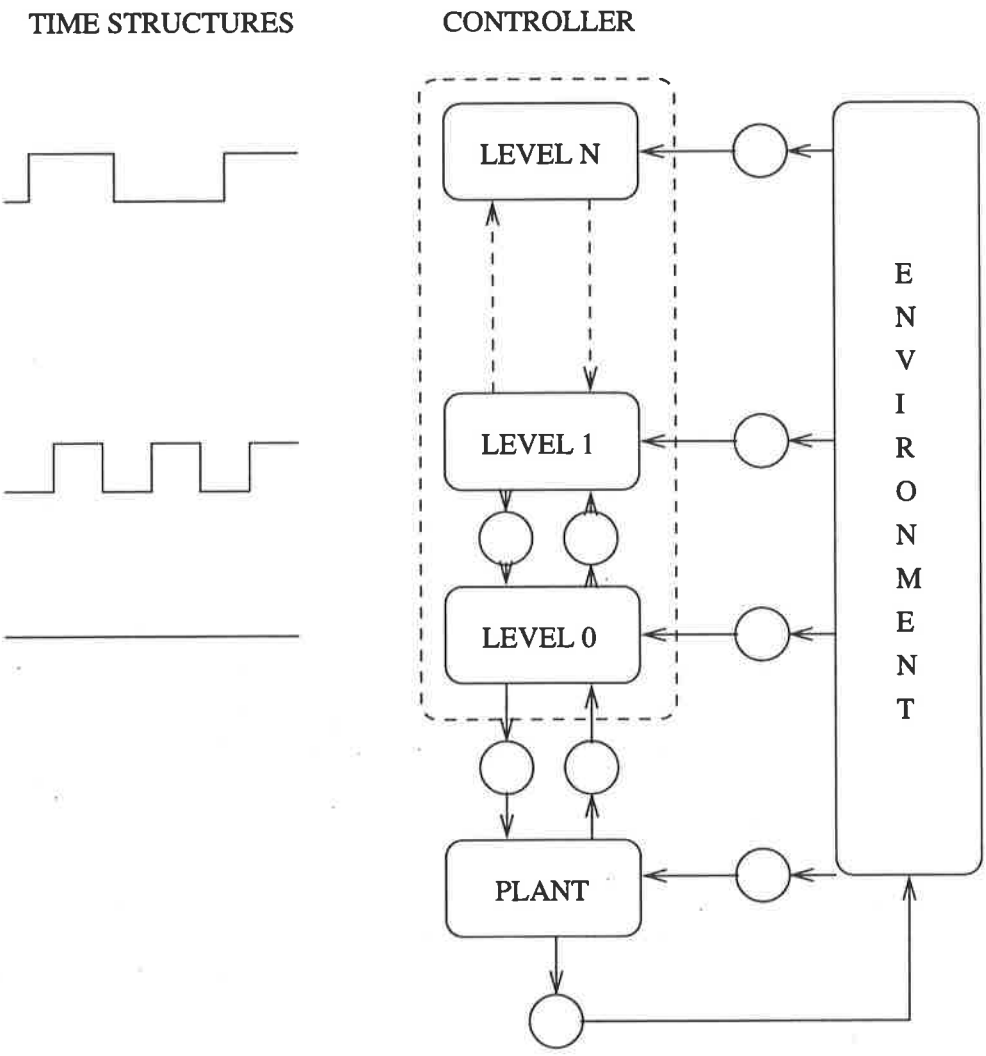

Figure 1: Abstraction hierarchy

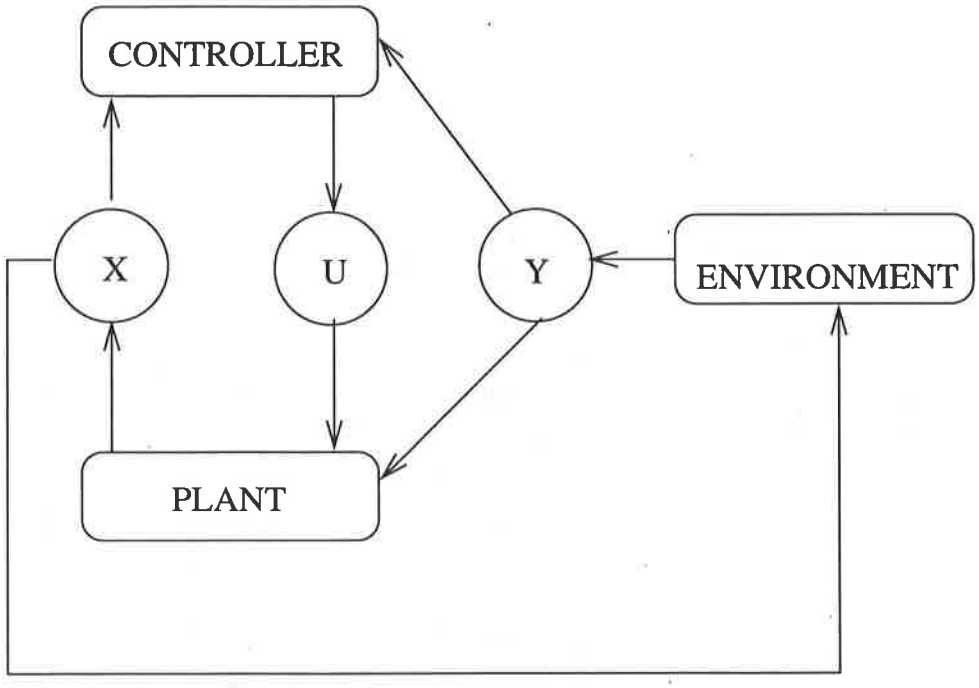

Figure 2: A robotic system

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 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).

8 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 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.

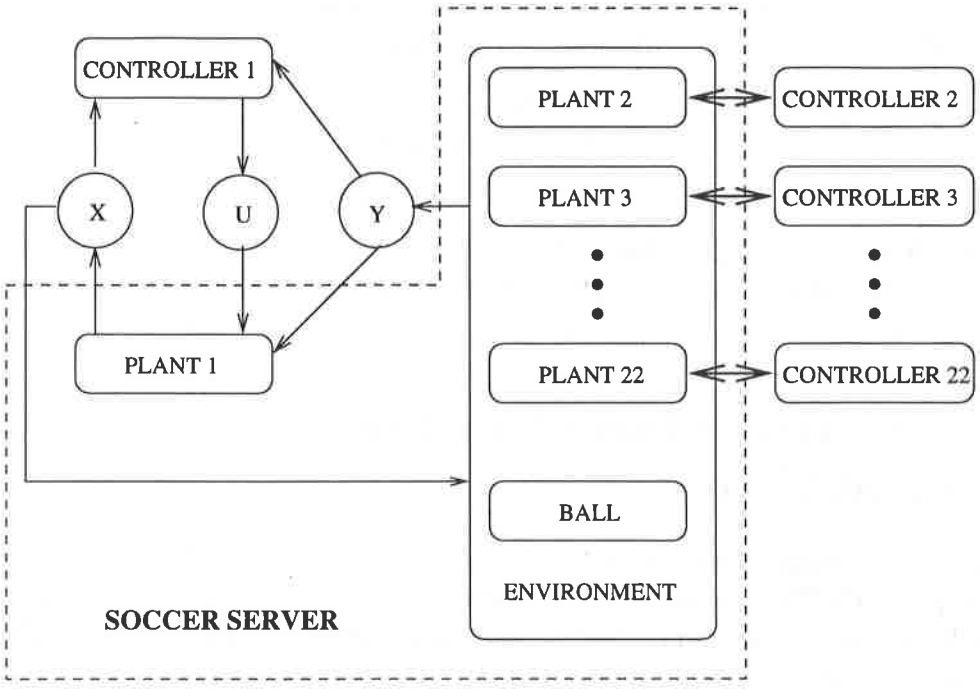

Figure 3: The soccer-playing softbot system

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*

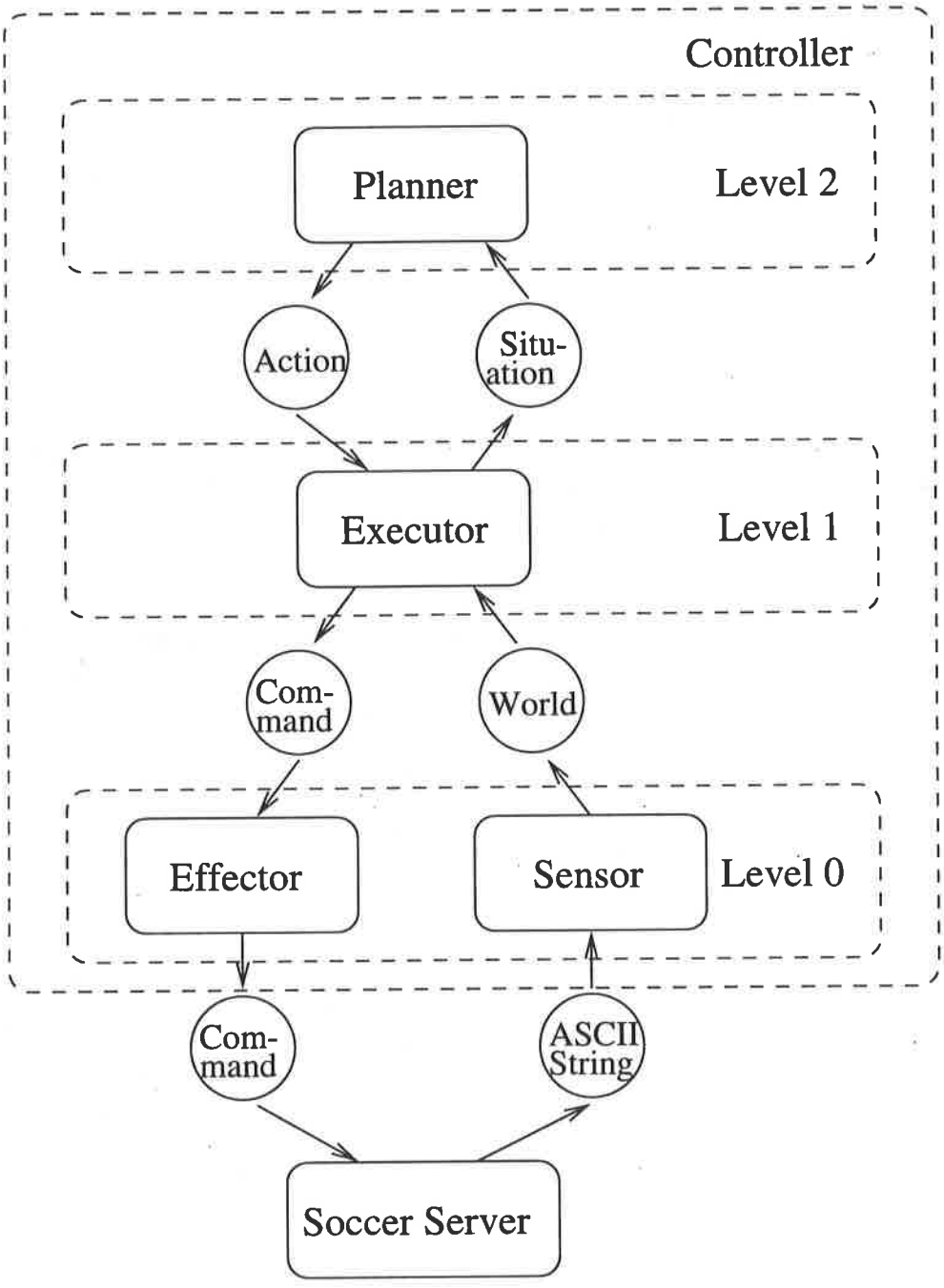

Figure 4: The soccer-playing controller hierarchy

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.

9 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 [16].

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 [17].

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.

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.

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	-

10 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.

11 Conclusions

We have described the motivation, some results and some current directions of a long-term project intended to develop a new approach to the specification, design, implementation and evaluation of robotic systems. The practical result is a new methodology for building intelligent, perceptual systems.

Acknowledgments

We wish to thank Ying Zhang for valuable discussions and suggestions. We are also grateful to Rod Barman, Cullen Jennings, Stewart Kingdon, Jim Little, Valerie McRae, Don Murray, Dinesh Pai, and Michael Sahota for help with this. This work is supported, in part, by the Natural Sciences and Engineering Research Council of Canada and the Institute for Robotics and Intelligent Systems Network of Centres of Excellence.

Bibliography

[1] R. L. Andersson. *A Robot Ping-Pong Player: Experiment in Real-Time Intelligent Control.* MIT Press, Cambridge, MA, 1988.

[2] D. H. Ballard. Reference frames for active vision. In *Proceedings IJCAI-89*, pages 1635–1641, Detroit, MI, 1989.

[3] R. A. Barman, S. J. Kingdon, J. J. Little, A. K. Mackworth, D. K. Pai, M. Sahota, H. Wilkinson, and Y. Zhang. DYNAMO: Real-time experiments with multiple mobile robots. In *Intelligent Vehicles Symposium*, pages 261–266, Tokyo, July 1993.

[4] R. A. Brooks. Intelligence without reason. In *IJCAI-91*, pages 569–595, Sydney, Australia, August 1991.

[5] D. Chapman. Vision instruction and action. Technical Report MIT AI TR-1085, MIT, Cambridge, MA, June 1990.

[6] I. D. Horswill and R. A. Brooks. Situated vision in a dynamic world: Chasing objects. In *AAAI-88*, pages 796–800, St. Paul, MN, 1988.

[7] J. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Stanford University, Stanford, CA, 1990.

[8] 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.

[9] A. K. Mackworth. Quick and clean: Constraint-based vision for situated robots. In *IEEE Int'l. Conf. on Image Processing*, pages 789–792, Lausanne, Switzerland, September 1996.

[10] S. J. Rosenschein and L. P. Kaelbling. The synthesis of machines with provable epistemic properties. In Joseph Halpern, editor, *Proc. Conf. on Theoretical Aspects of Reasoning about Knowledge*, pages 83–98. Morgan Kaufmann, Los Altos, CA, 1986.

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

[12] T. Winograd and F. Flores. *Understanding Computers and Cognition.* Addison-Wesley, Reading, MA, 1986.

[13] Y. Zhang and A. K. Mackworth. Specification and verification of constraint-based dynamic systems. In A. Borning, editor, *Principles and Practice of Constraint Programming*, number 874 in Lecture Notes in Computer Science, pages 229 – 242. Springer-Verlag, 1994.

[14] Y. Zhang and A. K. Mackworth. Will the robot do the right thing? In *Proc. Artificial Intelligence 94*, pages 255 – 262, Banff, Alberta, May 1994.

[15] Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems. *Theoretical Computer Science*, 138:211 – 239, 1995.

[16] Y. Zhang and A. K. Mackworth. Constraint programming in constraint nets. In V. Saraswat and P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming*, chapter 3, pages 49 – 68. The MIT Press, Cambridge, MA, 1995.

[17] Y. 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.

[18] Yu Zhang and A. K. Mackworth. A constraint-based controller for soccer-playing robots. In *IROS'98*, Victoria, BC, October 1998. To appear.

[19] Yu Zhang and A. K. Mackworth. A multi-level constraint-based controller for the Dynamo98 robot soccer team. In *RoboCu98*, Paris, France, July 1998. Springer-Verlag LNCS. To appear.

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

[21] Hiroaki Kitano *et al.*. RoboCup. Available at http://www.robocup.org

[22] 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.