Constraint-Based Agents: The ABC's of CBA's

Alan K. Mackworth

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

Abstract. The Constraint-Based Agent (CBA) framework is a set of tools for designing, simulating, building, verifying, optimizing, learning and debugging controllers for agents embedded in an active environment. The agent and the environment are modelled symmetrically as, possibly hybrid, dynamical systems in Constraint Nets, as developed by Zhang and Mackworth. This paper is a tutorial overview of the development and application of the CBA framework, emphasizing the important special case where the agent is an online constraint-satisfying device. Here it is often possible to verify complex agents as obeying real-time temporal constraint specifications and, sometimes, to synthesize controllers automatically. The CBA framework demonstrates the power of viewing constraint programming as the creation of online constraint-solvers for dynamic constraints.

1 Introduction

Constraint programming has evolved several powerful frameworks for building problem-solvers as constraint-satisfying devices. Primarily, these devices are offline problem-solvers. For example, the Constraint Satisfaction Problem (CSP) paradigm has evolved and matured over the last twenty-five years. The algorithms developed in the CSP paradigm were made more available and more useful when they were incorporated into the Constraint Programming (CP) language paradigms. Despite this success, however, a major challenge still facing the constraint research community is to develop useful theoretical and practical tools for the constraint-based design of embedded intelligent systems. Many applications require us to develop online constraint-satisfying systems that function in a dynamic, coupled environment [6]. An archetypal example of an application in this class is the design of controllers for sensory-based robots [13, 10, 7]. If we examine this problem we see that almost all the tools developed to date in the CSP and CP paradigms are inadequate for the task, despite the superficial attraction of the constraint-based approach. The fundamental difficulty is that, for the most part, the CSP and CP paradigms still presume a disembodied, offline model of computation.

Consider an agent coupled to its active environment as shown in Figure 1. Each is an open dynamic system in its own right, acting on, and reacting to, the other. The coupled pair form a closed system that evolves over time.

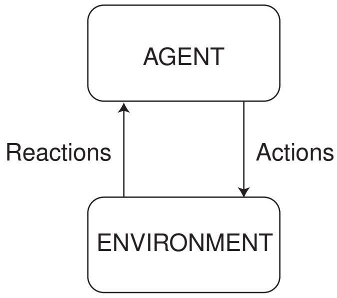

Fig. 1. An agent interacting with its environment

To deal with such embedded applications, we must radically shift our perspective on constraint satisfaction from the offline model in which a solution is a function of pre-given static inputs to an online model where a solution is a temporal trace of values, a transduction of the input trace over time. Values in the input trace may depend on earlier values in the output trace. In fact, the input trace for the agent is itself a mapping of its output trace, representing the dynamics of the environment, as shown in Figure 1.

Intelligent systems embedded as controllers in real or virtual systems must be designed in an online model based on various time structures: continuous, discrete and event-based. The requisite online computations, or transductions, are to be performed over various type structures including continuous and discrete domains. These hybrid systems require new models of computation, constraint satisfaction and constraint programming. To this end, we have defined constraint satisfaction as a dynamic system process that approaches asymptotically the solution set of the given, possibly time-varying, constraints [12]. Under this view, constraint programming is the creation of a dynamic system with the required property.

In this paper I present a tutorial overview of our approach, called Constraint-Based Agents (CBA), the ABC's of CBA's, if you like. The CBA model consists, at its simplest, of a symmetrical coupling of an agent and its active environment. As we'll see later, we say the agent is constraint-based if its behaviour satisfies a specification in a well-defined constraint-based temporal logic.

2 Agents in the World

The most obvious artificial agents in the world are robots. But the CBA approach applies equally to embedded devices, pure software agents and natural animate agents. There are many ways of using a CBA model, including the embedded mode, simulation mode, verification mode, optimization mode, learning mode and design mode [8, p. 449]. The agent design problem is formidable, regardless of whether the agent is designed or modified by a human, by nature (evolution), by another agent (bootstrapping), or by itself (learning). An agent is, typically, a hybrid intelligent system, consisting of a controller coupled to its body as shown in Figure 2.

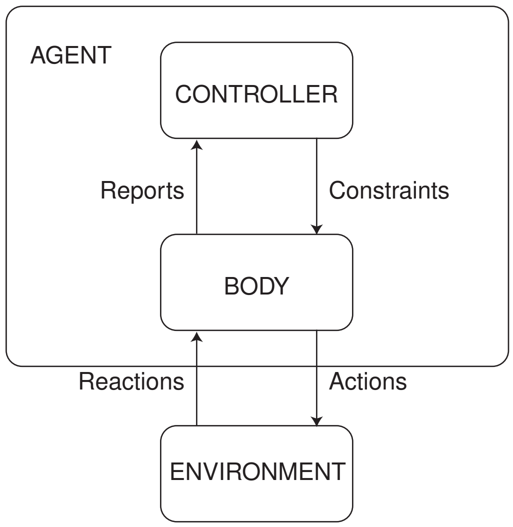

Fig. 2. The structure of a constraint-based agent system

The controller and the body 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 body, and through it, the state of the environment.

Parenthetically, the ‘body’ of an agent is simply the direct interface of the agent to its environment. The body executes actions in the environment, senses state in the environment (which may well cause state changes in the body) and reports to the controller. In the case of a robotic agent the body consists of one or more physical systems but in the case of an embedded software agent, the body is simply the software module that directly interfaces to the virtual or physical environment. Control theorists typically call the body the ‘plant’. Some models do not differentiate between the body and the environment; we prefer to make that differentiation, based on the distinction between what is directly, and what is indirectly, controlled.

Agent design methodologies are evolving dialectically [4]. The symbolic methods of 'Good Old Fashioned Artificial Intelligence and Robotics' (GOFAIR) constitute the original thesis. The antithesis is reactive 'Insect AI' and control theory. The emerging synthesis, Situated Agents, has promise, but needs formal rigor and practical tools [9, 3, 2, 4, 13, 7].

In 1992, I proposed robot soccer as a grand challenge problem [4] since it has the task characteristics that force us to confront the fundamental issues of agent design in a practical way for a perceptual, collaborative, real-time task with clear performance criteria. At the same time, I described the first system for playing robot soccer. Since then it has been a very productive testbed both for our laboratory [1, 10, 5, 16, 18, 17, 20] and for many other groups around the world, stimulating research toward the goal of building perceptual agents.

3 The Constraint Net Model

The Constraint Net (CN) model [14] was developed by Ying Zhang and Mackworth as a model for building hybrid intelligent systems as Situated Agents. In CN, an agent system is modelled formally as a symmetrical coupling of an agent with its environment. Even though a agent system is, typically, a hybrid dynamic system, its CN model is unitary. Most other agent and robot design methodologies use hybrid models of hybrid systems, awkwardly combining offline computational models of high-level perception, reasoning and planning with online models of low-level sensing and control.

CN is a model for agent 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 [14]. In applying it to an agent operating in a given environment, one separately specifies the behaviour of the agent body, the agent 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 agents. It can be seen as a formal development of Brooks' subsumption architecture [2] that enhances its reliability, modularity and scalability while avoiding the limitations of the augmented finite state machine approach, combining proactivity with reactivity.

An agent situated in an environment can be modelled as three machines: the agent body, the agent controller and the environment, as shown above in Figure 2. Each can be modelled separately as a dynamical system by specifying a CN with identified input and output ports. The agent is modelled as a CN consisting of a coupling of its body CN and its controller CN by identifying corresponding input and output ports. Similarly the agent CN is coupled to the environment CN to form a closed agent-environment CN, as shown above in Figure 1

The CN model is realized as an online dataflow-like distributed programming language with a formal algebraic denotational semantics and a specification language, a real-time temporal logic, that allows the designer to specify and prove properties of the situated agent by proving them of the agent-environment CN. We have shown how to specify, design, verify and implement systems for a robot that can track other robots [11], a robot that can escape from mazes and a two-handed robot that assembles objects [13], an elevator system [19] and a car-like robot that can plan and execute paths under non-holonomic constraints [16].

Although CN can carry out traditional symbolic computation online, 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, for controller synthesis and verification. 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 agent do the right thing?" [13] is answered positively if we can:

1. model the coupled agent 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 [16] and Timed $\forall$-automata [12], have been developed with appropriate theorem-proving and model-checking techniques for verifying systems.

In [8, Chapter 12] we describe how to build a situated robot controller using CN as realized in a logic program.

4 Constraint-Satisfying Agents

Many agents can be designed as online constraint-satisfying devices [12, 15, 16]. A robot in this restricted scheme can be verified more easily. Moreover, given a constraint-based specification and a model of the body and the environment, automatic synthesis of a correct constraint-satisfying controller sometimes becomes feasible, as shown for a simple goal-scoring robot in [16].

As a simple example, in Figure 2 suppose the CONTROLLER is a thermostat turning on or off a furnace, the BODY, that is heating the ENVIRONMENT. The goal of the system is to make the temperature of of the ENVIRONMENT, T_E, equal to a desired temperature, T_D. In other words the CONTROLLER of the AGENT is trying to solve the constraint $T_E(t) = T_D(t)$. One version of CONTROLLER correctness is established if we can prove that the (thermal) dynamics of the coupled AGENT-ENVIRONMENT system satisfy the temporal logic formula $\diamond\Box|T_E - T_D| < \epsilon$ where $\diamond$ can be read as 'eventually' and $\Box$ can be read as 'always' In other words, the system will, no matter how disturbed, eventually enter, and remain within, an ϵ-neighborhood of the solution manifold of the constraint. A less restrictive form of correctness corresponds to the specification $\Box\diamond|T_E - T_D| < \epsilon$ which is to say that the system will always return, asymptotically, to the constraint solution manifold if it should happen to leave it.

A constraint is simply a relation on the phase space of the agent system, which is the product of the controller, body 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 or a symbolic technique.

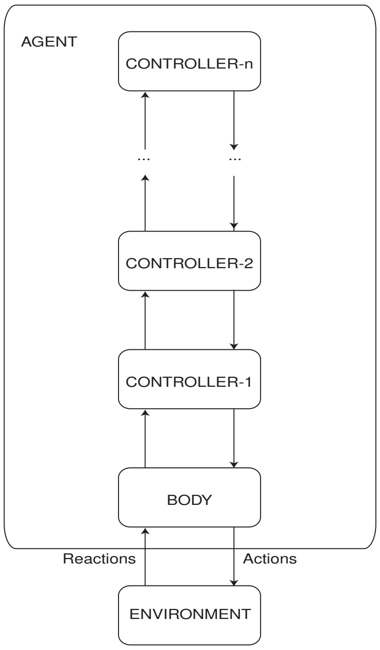

Fig. 3. A hierarchical agent controller

A constraint-satisfying controller may be *hierarchical* with several layers of controller above the body, as shown in Figure 3. In this case, each layer must satisfy the constraints, defined on its state variables, appropriate to the layer, as, typically, set by the layer above. The layers below each layer present to that layer as a virtual agent body in a suitably abstract state space [16, 17]. The lower layers are, typically, reactive and synchronous (or in continous time) on continuous state spaces; the upper layers are more deliberative and asynchronous (or event-triggered) in symbolic, discrete spaces.

A typical layer in a hierarchical controller is shown in Figure 4.

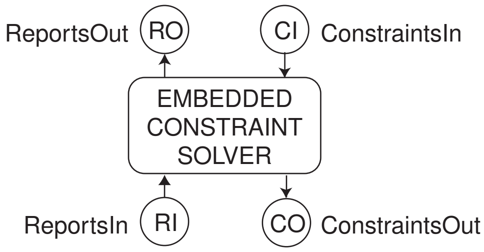

Fig. 4. A layer in a constraint-based controller

Each layer has two external inputs: the trace of constraint requests coming from above *ConstraintsIn* (CI) and the reports coming from below *ReportsIn* (RI). Its two outputs are its reports to the level above *ReportsOut* (RO) and its constraint requests to the level below *ConstraintsOut* (CO). These traces arise from *causal transductions* of the external inputs:

$$CO = C_t(CI, RI) \tag{1}$$

$$RO = R_t(CI, RI) \tag{2}$$

If the constraint-solver can be represented as a state-based solver then the layer may be represented as shown in Figure 5.

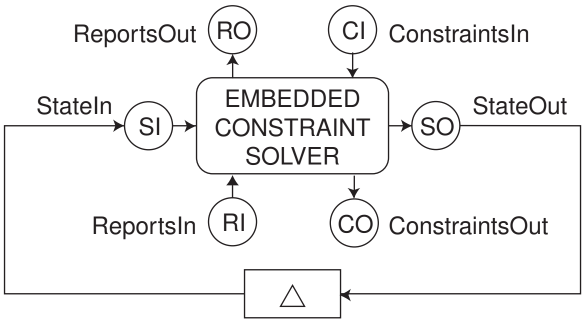

Fig. 5. A layer with state in a constraint-based controller

Here, for simplicity, a discrete-time state-based layer is shown. It produces an extra output *StateOut* (SO) that is consumed as an extra input *StateIn* (SI) after a unit delay (Δ). In this case the behaviour of the layer may be represented by computing the values of the three outputs as *transliterations* (functions) of the current values of the three inputs:

$$co(t) = c_f(ci(t), ri(t), si(t)) \tag{3}$$
$$ro(t) = r_f(ci(t), ri(t), si(t)) \tag{4}$$
$$so(t) = s_f(ci(t), ri(t), si(t)) \tag{5}$$
$$si(t+1) = so(t) \tag{6}$$

5 Robot Soccer Players

The CBA framework has also been motivated, developed and tested by application to the challenge of designing, building and verifying controllers with perceptual 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 [1], including playing soccer [10] using a 2-layer deliberative/reactive controller architecture.

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 as its head and a trinocular stereo camera in its base. As a illustration of these ideas, consider the task for Spinoza of repeatedly finding, tracking, chasing and kicking a soccer ball, using the pan-tilt camera. After locating the moving ball Spinoza is required to track it, move to within striking distance of the ball and strike it. 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 base rotation. A hierarchical constraint-based active-vision controller, using prioritized constraints and constraint arbiters, can be specified for Spinoza that will, repeatedly, achieve and maintain (or re-achieve) 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.

Yu Zhang and Mackworth have extended these ideas to build 3-layer constraint-satisfying controllers for a complete soccer team [20]. The controllers for our softbot soccer team, UBC Dynamo98, are modelled in CN and implemented in Java, using the Java Beans architecture [17]. They control the soccer players' bodies in the Soccer Server developed by Noda Itsuki for RoboCup. These experiments provide evidence that the constraint-based CN approach is a clean and practical design framework for perceptual robots.

6 Conclusions

The Constraint-Based Agent approach is a framework for the specification, design, analysis, implementation and validation of artificial and natural agent systems. It requires a new model of online and embedded computation for Constraint Programming such as Constraint Nets.

7 Acknowledgments

I am most grateful to Ying Zhang and Yu Zhang for our collaborations. I also thank Rod Barman, Cullen Jennings, Stewart Kingdon, Jim Little, Valerie McRae, Don Murray, Dinesh Pai, David Poole, Michael Sahota, and Vlad Tucakov 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.

References

1. 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.
2. R. A. Brooks. Intelligence without reason. In *IJCAI-91*, pages 569–595, Sydney, Australia, Aug. 1991.
3. J. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Stanford University, Stanford, CA, 1990.
4. 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.
5. 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, Sept. 1996.
6. A. K. Mackworth. Constraint-based design of embedded intelligent systems. *Constraints*, 2(1):83–86, 1997.
7. A. K. Mackworth. The dynamics of intelligence: Constraint-satisfying hybrid systems for perceptual agents. In *Hybrid Systems and AI: Modeling, Analysis and Control of Discrete and Continuous Systems*, number SS-99-05 in AAAI, Spring Symposium Series, pages 210–214, Stanford, CA, Mar. 1999.

8. D. L. Poole, A. K. Mackworth, and R. G. Goebel. *Computational Intelligence: A Logical Approach.* Oxford University Press, New York, 1998.
9. 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.
10. M. Sahota and A. K. Mackworth. Can situated robots play soccer? In *Proc. Artificial Intelligence 94*, pages 249 – 254, Banff, Alberta, May 1994.
11. Y. Zhang and A. K. Mackworth. Modeling behavioral dynamics in discrete robotic systems with logical concurrent objects. In S. G. Tzafestas and J. C. Gentina, editors, *Robotics and Flexible Manufacturing Systems*, pages 187–196. Elsevier Science Publishers B.V., 1992.
12. 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.
13. 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.
14. Y. Zhang and A. K. Mackworth. Constraint Nets: A semantic model for hybrid dynamic systems. *Theoretical Computer Science*, 138:211 – 239, 1995.
15. 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.
16. 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.
17. Y. Zhang and A. K. Mackworth. A constraint-based controller for soccer-playing robots. In *Proceedings of IROS '98*, pages 1290 – 1295, Victoria, BC, Canada, Oct. 1998.
18. Y. Zhang and A. K. Mackworth. Using reactive deliberation for real-time control of soccer-playing robots. In H. Kitano, editor, *RoboCup-97: Robot Soccer World Cup 1*, pages 508–512. Springer-Verlag, Aug. 1998.
19. Y. Zhang and A. K. Mackworth. Modelling and analysis of hybrid systems: An elevator case study. In H. Levesque and F. Pirri, editors, *Logical Foundations for Cognitive Agents*, pages 370–396. Springer, Berlin, 1999.
20. Y. Zhang and A. K. Mackworth. A multi-level constraint-based controller for the Dynamo98 robot soccer team. In Minoru Asada and Hiroaki Kitano, editor, *RoboCup-98: Robot Soccer World Cup II*, pages 402–409. Springer, 1999.