REPRINTED FROM:

Robotics and Flexible Manufacturing Systems

Selected and revised papers from the IMACS 13th World Congress, Dublin, Ireland, July 1991, and the IMACS Conference on Modelling and Control of Technological Systems, Lille, France, May 1991

edited by

Jean-Claude GENTINA
Ecole Centrale de Lille
Villeneuve d'Ascq, France

Spyros G. TZAFESTAS
Computer Engineering Division
National Technical University of Athens
Athens, Greece

1992

NORTH-HOLLAND
AMSTERDAM • LONDON • NEW YORK • TOKYO

Robotics and Flexible Manufacturing Systems
J.C. Gentina and S.G. Tzafestas (Editors)
Elsevier Science Publishers B.V. (North-Holland)
© 1992. IMACS. All rights reserved.

Modeling Behavioral Dynamics in Discrete Robotic Systems with Logical Concurrent Objects

Ying Zhang and Alan K. Mackworth *
Department of Computer Science
University of British Columbia
Vancouver, B.C. Canada
E-mail: zhang@cs.ubc.ca, mack@cs.ubc.ca

Abstract

Robots are generally composed of multiple sensors, actuators and electromechanical components. Robots are reactive as well as purposive systems, closely coupled with their environments; they must deal with inconsistent, incomplete and delayed information from various sources. Such systems are usually complex, hierarchical and physically distributed. Each component functions according to its own dynamics. Even though a lot of work has been done on the design of control systems, there is little understanding of the overall behavioral dynamics, which is emergent from the interactions among various components and their environment. We have proposed a mathematical framework of behavioral dynamics called the Constraint Net model (CN) [1]. CN provides a mathematical semantics for real-time programming languages. In this paper, we give an overall introduction to the Constraint Net model and then show that discrete constraint nets can be simulated by logical concurrent objects. Logical concurrent objects support multi-agent communication architectures, stream representations and direct broadcast, which provide a formal and executable specification for the behavioral dynamics in discrete robotic systems.

1 Motivation and Introduction

The most important characteristic of animals or robots is to react appropriately to their environment. A *robot* is an integration of the *control* with the *plant*. We have called a robot an *open robot* [2], if the robot can work in unstructured and unpredictable environments, deal with inconsistent or incomplete information, and react to the environment in real-time. We call this kind of robot an open robot because its characteristics are consistent with the characteristics of an *open system* [3]. Any living system is an open system [4]; being open is the most important characteristic of a living system. As an

*Shell Canada Fellow, Canadian Institute for Advanced Research open system, it can interact with the environment, actively obtain information from the environment, and adaptively achieve balance within the environment. Such robots are useful for space applications, undersea exploration and forest harvesting.

The most important characteristic of open robot control systems is their distributed and asynchronous architecture. Centralized control or synchronized timing structures have inherent limitations for multi-sensory and multi-actuator systems [2], while distributed control and event-driven structures make the overall dynamics extremely complicated and unpredictable [3, 5]. There is a strong need for understanding, modeling and analysis of open robots.

A *robotic system* is an integration of an open robot with its environment. We have developed a mathematical framework called the Constraint Net model (CN) for modeling behavioral dynamics in robotic systems [1]. CN is a real-time model, in which both data- and control-flow can be represented. CN, in general, models distributed and asynchronous systems, while centralized or synchronous control structures are special cases. CN provides a unified framework in which the environment as well as the control and the plant can be modeled and the overall dynamics of the system can be analyzed. CN supports multiple levels of abstraction so that a system can be modeled hierarchically and verified incrementally.

CN provides a mathematical semantics for real-time programming languages. Most widely used robot programming languages nowadays are not suitable for modeling behavioral dynamics. They lack capabilities for describing parallel actions and sensorimotor coordination. In [6], we have used parallel C++, an integration of C++ with parallel C, for modeling and implementing the behavior of a robot arm with six joints. However, this kind of modeling is too low-level to analyze the behavior at the design stage. Brooks [7] designed and implemented the Behavior Language, a rule-based real-time parallel robot programming language written in Lisp. However, the semantics of that language is not clear, which again makes it hard to analyze programs and to prove correctness before implementation. Models like Petri Nets [8] or Bond Graphs [9] are useful mainly for system modeling and analysis. What we seek is an executable, yet formal, specification, with clear and simple semantics for implementing, as well as modeling and analyzing behavioral dynamics.

Concurrent logic programming languages like Strand [10], or Concurrent Constraint Logic Programming languages [11] in general, support multi-agent communication architectures, stream representations with infinite fan-outs and direct broadcast. This family of languages has been suggested as a good candidate for open systems [12]. An object-oriented framework provides modularity and hierarchical architecture, which are essential for modeling complicated behaviors.

We develop an object-oriented framework on top of Strand to simulate constraint nets with discrete reference time structures. This executable model can be used to prototype the design and to assist the verification of robotic systems.

In this paper, we begin with the discussion of behavioral dynamics in robotic systems, then present the modeling of behavioral dynamics with constraint nets, and finally show how to simulate discrete constraint nets with logical concurrent objects.

2 Behavioral Dynamics in Robotic Systems

The study of dynamics is concerned with how things change over time [5]. The study of systems is concerned with how a system's overall behavior is generated through the interactions of its components. Robots are generally composed of multiple sensors, actuators and electromechanical components. Robots are reactive as well as purposive systems, closely coupled with their environments; they must deal with inconsistent, incomplete and delayed information from various sources. Such systems are usually complex, hierarchical and physically distributed. Each component functions according to its own dynamics. The overall behavior of a robot cannot be determined by any of its single component. Rather, it is emergent from the coupling of the dynamics of its various parts and its interaction with the environment.

2.1 General Conceptions

For our purposes, a *variable* is a function of time. For instance, positions, velocities, internal states, forces and control signals are modeled as variables. Variables may be related to each other. The possible class of relations in physical systems is abstracted as *transductions.* A transduction is a state-determined transformational process whose output signal at any time is dependent on the input signals prior to that time. For instance, integration, delay and finite state automata can be modeled as transductions. Formally, a transduction is a function from a vector of input variables to an output variable, whose value at any time depends only on those input values before or at that time. We define a *reference time* as a global time of a system, and a *clock* as a special kind of variable, which represent the local time or event structures of transductions (see Figure 1, where a transition from 0 to 1 or 1 to 0 defines an event). Some transductions are triggered

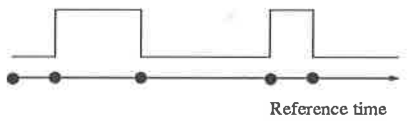

Figure 1: A clock

only at "points" created by clocks, while others may work at all times. A *dynamic system* is a system defined on a reference time, which consists of a set of transductions and a set of variables. Some variables in the system can be clocks. Systems modeled with clocks can represent discrete control of asynchronous as well as synchronous interactions between various components. *Behavioral dynamics* in robotic systems captures the the dynamic interactions between an open robot and its environment.

2.2 An Example

Consider the robotic system shown in Figure 2, where robot R always tries to track object A and to avoid B. R consists of a sensor which can detect the distances and directions of

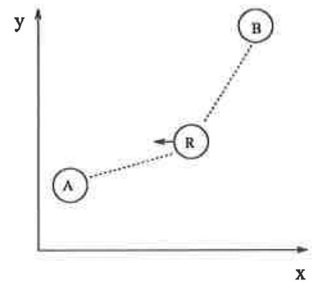

Figure 2: A robot world

objects A and B up to a maximum range, and a steerable motor which can move R in one of the four cardinal directions. A command to the motor is the desired velocity vector, whose value is related to the sensed signal. We are interested in the following aspects of the system:

- What is the simplest design for the controller of R?
- Is the controller robust, i.e. what ranges of delay or uncertainty can be tolerant in the design?
- What restrictions should be imposed on the environment to show that R can achieve its goal?

Even for a simple system like this, these questions are hard, if not impossible, to answer without modeling and simulation.

3 Modeling Behavioral Dynamics with Constraint Nets

3.1 Introduction to Constraint Nets

A *constraint net* models a dynamic system. Syntactically, a constraint net is a triple $CN \equiv \langle Lc, Td, Cn \rangle$ where Lc is a set of *locations* which denote variables, Td is a set of *transductions* and Cn is a set of directed *connections* between locations and transductions, with the restriction that (1) there is at most one connection pointing to each location, (2) each *port* of a transduction connects to a unique location, (3) no location is isolated. A location is an *input* if there is no connection pointing to it, otherwise it is an *output*.

A constraint net is *closed* iff there is no input location, otherwise it is *open*. An open robot can be modeled by an open constraint net, where the open locations denote the environment variables.

There are some basic operations that can be applied to one or more constraint nets to construct a new constraint net. These operations include *composition*, which combines two nets into one; *equalization*, which identifies two or more locations, with the restriction that only one of them is an output location, into one; and *reconnection*, which changes the connections between locations and transductions. With these operations, a complex constraint net can be built out of simple ones. We define a *module* to be a constraint net associated with a subset of its output locations, which, with its input locations, defines the *interface* of the module. A constraint net can be hierarchically constructed with modules.

An equivalent representation of a constraint net is a set of equations, where each left-hand side is an individual output location and each right-hand side is an expression composed of transductions and locations. The semantics of a constraint net is defined as the least fixpoint of the set of equations, which is called the *trajectory* of the dynamic system. Formally, the trajectory of a dynamic system modeled by a constraint net is a mapping from each output location to a transduction, which is a function from the vector of input variables in the input locations to the output variable in this location. For a complex system with multiple components, the behavior of the whole system can be obtained by the behaviors of its components and their connections.

In a word, the semantics of a constraint net corresponds to the trajectory of the system being modeled, which determines the behavior of that system. We call this the Constraint Net model, since a constraint net behaves as if it is solving the constraints imposed by various parts of the system. In fact, most relaxation algorithms (such as Newton's method and various neural network algorithms) can be implemented on this model directly.

A *robotic system* is a closed dynamic system which includes the control, the plant and the environment (see Figure 3, where circles are vectors of locations and boxes are or modules). The overall behavior of the system is determined by the fixpoint of the dynamic

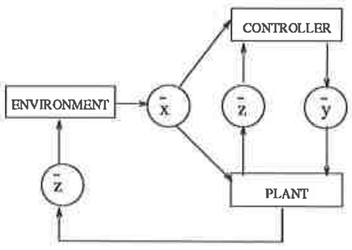

Figure 3: A robotic system modeled by a constraint net

behavior of each component and their interactions. The design task for a robotic system can be characterized as: given the models of some of the parts of the system, design the rest of the system so that the desired behaviors can be achieved.

3.2 The Tracking Robot Example

We can model the tracking robot in the previous section by the constraint net shown in Figure 4, where `Plant` is an integrator from velocity to position, `Sensor` is a transduction from the current position variable and the environment variables (positions of A and B) to the sensed signals, `Controller` is a module whose inputs are the sensed signals and output is the desired velocity. The controller module can be further decomposed into two layers, where the lower layer is a transduction for avoiding objects and the higher layer is a transduction for tracking objects. The final command is obtained via the `Arbiter`. One possible arbiter is a so-called *priority gate*: the lower layer command has a higher execution priority. In this example, it ensures that the robot tracks A only if it will not be caught by B.

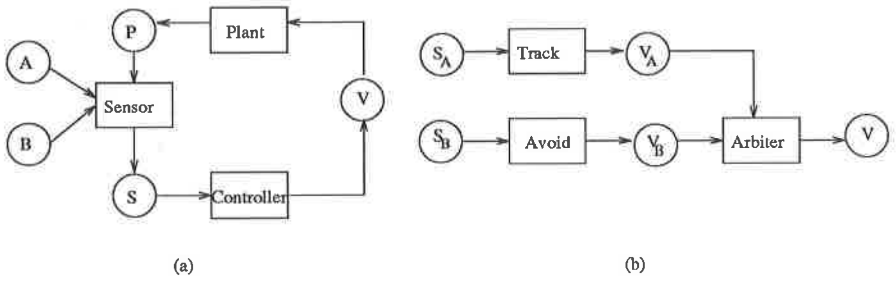

Figure 4: (a) The Constraint Net model of the open robot
(b) Further decomposition of the controller module

4 Simulating Discrete Constraint Nets with Logical Concurrent Objects

4.1 Logical Concurrent Objects

Concurrent Constraint Logic Programming [11] provides some unique high-level primitives for process communication and synchronization, among which are direct broadcast, stream representations with incomplete messages and infinite fan-out, all under a uniform declarative semantics. Languages of a similar style include FCP, Parlog, FGHC and Strand [10]. This family of languages has been suggested as a good candidate for open systems [12]. An object-oriented framework provides modularity and hierarchical architecture, which are essential for modeling complicated behaviors.

We developed an object-oriented framework on top of Strand, which can be used to simulate *discrete* constraint nets, systems whose reference times are discrete. We formulate discrete constraint nets as an extension of the distributed constraint logic paradigm. The extension emphasizes the following aspects:

- logical concurrent objects as well as explicit port-to-port communications between objects can be defined and manipulated explicitly;
- transductions are represented by logical concurrent objects and variables are represented by streams.

In general, a system is constructed using a set of transductions, each of which has its own internal state, as well as a set of input ports and an output port. The behavior of a transduction is defined by a set of transitions. Each transition maps the input and current internal state into a set of actions. More specifically, a distributed constraint logic programming language is extended with the following primitives:

- the definition of transductions:

  ```
  class(transduction, [in1,in2,...], [state1,state2,...], out).

  transduction::init => A1, A2, ... An

  transduction#in1(Message1)#in2(Message2)... =>
        C1, C2, ...? A1, A2, ... An
  ```

  where `Ci` are guarded constraints and `Ai` are actions. These actions include sending messages to its output port:

  ```
  out <: Message
  ```

  and updating its internal state:

  ```
  new statei := NewStatei
  ```

- the use of transductions:

  ```
  transduction(OutVar, InitState, InVars)
  ```

  where `OutVar` is the output variable of the transduction, `InitState` is the initial state vector and `InVars` is the vector of input variables of the transduction.

4.2 The Tracking Robot Constraint Net Example

We can model the tracking robot using this framework. Let `delta` be the step size of the discrete reference time, `maxrange` be the range of the sensors, and `k` be a control parameter.

```
class(plant, [velocity], [x,y], position).
plant::init => position <: [x,y].
plant#velocity([Vx,Vy]) =>
     Px := x + delta * Vx, Py := y + delta * Vy,
     new x := Px, new y := Py,
     position <: [Px,Py].

class(sensor, [robot,object], [], signal).
sensor#robot([Rx,Ry])#object([Ox,Oy]) =>
     Dx := Ox - Rx, Dy := Oy - Ry,
     range(Dx,Dx1), range(Dy,Dy1),
     signal <: [Dx1,Dy1].
range(X,X1) => X >= maxrange ? X1 := 0.
range(X,X1) => X < -maxrange ? X1 := 0.
range(X,X1) => otherwise ? X1 := X.

class(track, [sensor], [], motor).
track#sensor([0,0]) => motor <: idle.
track#sensor([X,Y]) => X > 0, Y > 0, X > Y ?
     Vx := k * X, motor <: [Vx,0].
track#sensor([X,Y]) => X > 0, Y > 0, Y >= X ?
     Vy := k * Y, motor <: [0,Vy].
  ...
  ...

%(X0,Y0) is the initial position of the robot.
%A is the position variable of the object being tracked.
robot([X0,Y0],A) =>
     plant([V], [X0,Y0], P),
     sensor([P,A], [], S),
     track([S], [], V).
```

Figure 4.2 shows a trace of this robot for a particular `X0`, `Y0` and `A`.

Even though it is a simple system, the underlying dynamics can be very complex. It is interesting to see that the robot may never catch the object, if the maximum range of the sensor is limited, or the control parameter is small, or the object moves too fast. The dynamics gets more complicated if there are sensing or computation delays. We can model the `avoid` transduction similarly and integrate the `track` and the `avoid` with the `arbiter`.

```
class(arbiter, [in1,in2], [], out).
arbiter#in1(In1)#in2(In2) => In1 == idle ?
     out <: In2.
arbiter#in1(In1)#in2(In2) => otherwise ?
     out <: In1.
```

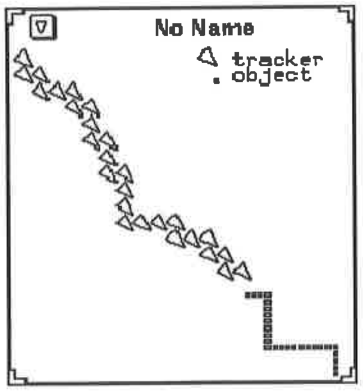

Figure 5: A trace of the tracking robot

```
sensors(A,B,P,SA,SB) =>
     sensor([P,A], [], SA),
     sensor([P,B], [], SB).

controller(SA,SB,V) =>
     track([SA], [], VA),
     avoid([SB], [], VB),
     arbiter([VB,VA], [], V).
```

We can further model delays and distributed clocks, so that the system can be tested under various assumptions.

5 Conclusion and Future Work

The behavioral dynamics of the interaction of an open robot and its environment can be very complex even for a simple system. The Constraint Nets model provides a unified framework for modeling, simulation and analysis of robotic systems. Logical concurrent objects can be used to simulate discrete constraint nets, therefore provide a formal and executable specification for the behavioral dynamics in discrete robotic systems.

We plan to further develop a visual programming and simulation environment, known as ALERTS — A Laboratory for Embedded Real Time Systems, based on the Constraint Net model. With ALERTS, a robotic system can be designed hierarchically, and simulated or verified incrementally.

Acknowledgements

We wish to thank Peter D. Lawrence for valuable suggestions. The first author is supported by the University Graduate Fellowship from University of British Columbia. This research was supported by the Natural Sciences and Engineering Research Council and the Institute for Robotics and Intelligent Systems.

References

[1] Ying Zhang and Alan K. Mackworth. Constraint nets — a model of behavioral dynamics in real-time robotic systems. In Preparation.

[2] Ying Zhang. Transputer-based behavioral module for multi-sensory robot control. In Mike Reeve and Steven Ericsson Zenith, editors, *Parallel Processing and Artificial Intelligence*, Communication Process Architecture. Wiley, 1989.

[3] B. A. Huberman. The ecology of computation. In B. A. Huberman, editor, *The Ecology of Computation.* Elsevier Science Publishers B.V.(North-Holland), 1988.

[4] James G. Miller. *Living Systems.* McGRAW-Hill Book Company, 1978.

[5] James T. Sandfur. *Discrete Dynamical Systems: Theory and Applications.* Clarendon Press, 1990.

[6] Ying Zhang. Object oriented modeling for sensor-guided real-time robot control. In Alan S. Wagner, editor, *Transputer Research and Applications 3.* IOS Press, 1990.

[7] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1), March 1986.

[8] Paul Freedman. Time, petri nets, and robotics. *IEEE Transactions on Robotics and Automation*, 7(4), August 1991.

[9] Dean C. Karnopp, Donald L. Margolis, and Ronald C. Rosenberg. *System Dynamics: A Unified Approach.* Wiley, 1990.

[10] Ian Foster and Steven Taylor. *Strand: New Concepts in Parallel Programming.* Prentice Hall, 1989.

[11] Vijay Anand Saraswat. Concurrent constraint programming languages. Technical report, Computer Science Department, Carnegie–Mellon University, 1989. Ph. D. thesis.

[12] Kenneth M. Kahn and Mark S. Miller. Language design and open systems. In B. A. Huberman, editor, *The Ecology of Computation.* Elsevier Science Publishers B.V.(North-Holland), 1988.