Planning and Acting in Real-Time

D.M. Lyons and A.J. Hendriks
Philips Laboratories
North American Philips Corporation
Briarcliff Manor, NY 10510

Invited paper. 13th IMACS World Congress on Computation and Applied Mathematics, July 22-26, 1991, Trinity College, Dublin, Ireland.

1 Introduction

Our work is directed at solving the kitting robot problem; this necessitates producing a robot system that can both plan *and* react in an uncertain and dynamic environment. We have developed a novel architecture that integrates planning and reaction by casting planning as the goal-directed adaptation of a reactive system (the reactor) [3]. This paper presents an overview of our methodology for characterizing the real-time properties of the reactor.

2 Planner-Reactor System

The reactor is a system, like those discussed by Brooks [1] and others, that contains a network of sensory-motor 'linkages' which we will call reactions. The reactor has several key properties. First, it can produce action at *any* time. Unlike a classical plan executor, a reactor is always actively inspecting the world, and will act should one of its reactions be triggered. Secondly, it is a *real-time* system; it needs to produce the appropriate action at the appropriate time. To make this possible, all the 'intelligence' in a reactor is encoded into reactions and cross-connections between them, and annotated with time-constraints.

The planner is completely separate from, and concurrent with, the reactor. It interacts with the reactor to produce appropriate behavior in the same fashion that the reactor interacts with the world. For more details of the planner-reactor system, see our companion paper on that topic in these proceedings.

3 The $\mathcal{RS}$ Model

We use a unique, process-based, formal model of computation, the $\mathcal{RS}$ model [2], to describe and analyze the reactor. A very brief introduction to this model is given here; see [3] for more details. An $\mathcal{RS}$ description of a process, or network of processes, is called a *schema*. For example, $\mathtt{Joint}_{j=v}$ denotes a process that is an instance of the schema $\mathtt{Joint}$ with parameter j set to value v (when unambiguous, we write $\mathtt{Joint}_v$). Processes can be defined in terms of networks of other processes, grounding out with a set of pre-defined, basic processes. Processes can be coupled into networks in two ways: they can communicate messages to one another via *communication ports*, or they can be composed together using several kinds of *process composition operators*. Process composition operators include: Network composition $(\mathtt{A},\mathtt{B})^{\mathtt{c}}$, Disabling composition $(\mathtt{A\#B})$, Enabling composition $\mathtt{A}\langle x\rangle\mathtt{:B}_x$, Sequential composition $\mathtt{A;B}$, and two recurrent compositions, Recurrent Synchronous $\mathtt{A}\langle x\rangle\ \mathtt{:;B}_x$, and Recurrent Asynchronous $\mathtt{A}\langle x\rangle\ \mathtt{::B}_x$. To represent how process networks evolve over time, as processes dynamically terminate or are created, we developed the evolves operator: process $\mathtt{A}$ evolves into process $\mathtt{B}$ under condition Ω if $\mathtt{A}$ becomes equal to $\mathtt{B}$ when condition Ω occurs; written $\mathtt{A} \xrightarrow{\Omega} \mathtt{B}$.

It is not possible to describe the behavior of a reactive system in any concise way without referring to the environment in which the system is active. Therefore, the appropriate unit to analyze is the system embedded or situated in its environment. Our approach in $\mathcal{RS}$ is to build world models from the same tools used to build plans — networks of concurrent processes.

<u>Temporal Specification.</u> We introduce the following operator to specify the temporal properties of processes: $\mathtt{A}\ @\ \mathtt{B}$ specifies that the processes $\mathtt{A}$ and $\mathtt{B}$ begin and execute for the same times (this is a very weak form of

a *specification* or *implementation* operator). The following temporal basic processes are introduced: $\mathtt{D}_t$ is a process that always executes for a time t; $\mathtt{T}_t$ is a process that always terminates at time t (if it was started before or at t); TERM is a process that executes for an arbitrary amount of time. The relationship between @ and the other operators can be characterized as follows:

$$\begin{array}{lcl} \mathtt{A@D}_a;\mathtt{B@D}_b & \Rightarrow & \mathtt{A;B@D}_{a+b} \\ (\mathtt{A@D}_a,\mathtt{B@D}_b) & \Rightarrow & (\mathtt{A,B})\mathtt{@D}_{max(a,b)} \\ (\mathtt{A@D}_a\#\mathtt{B@D}_b) & \Rightarrow & (\mathtt{A\#B})\mathtt{@D}_{min(a,b)} \end{array}$$

Because Enabling composition has conditional effects, its relationship with @ is beyond the scope of this paper. It is possible to use @ to describe a range of temporal constraints, e.g.:

$$\begin{array}{ll} duration(d): & \mathtt{A@D}_d \\ deadline(d): & \mathtt{A@(TERM\#D}_d) \\ interval(d1,d2): & \mathtt{A@(D}_{d1};(\mathtt{TERM\#D}_{d2})) \\ earliest\ start(t): & \mathtt{D}_t;\mathtt{TERM};\mathtt{A} \end{array}$$

We abbreviate the interval constraint $\mathtt{Int}_{d1,d2} = (\mathtt{D}_{d1};(\mathtt{TERM\#D}_{d2}))$.

4 Temporal Properties

The @ operator is used to characterize the real-time properties of the reactor. Below are examples of two such properties: a *control* property, the *timeliness* of a reaction, and a *scheduling* property, the *loading* of the reactor.

Within the reactor, plan segments and reactions are represented in the following, *situation-labeled* form[3]:

$$\mathtt{Pa} = \Big|_{s\in sit(a)} (\ \mathtt{CheckSit}_s : \mathtt{Action}_{a,s}\ \#\ I_{a,s}\ ) \tag{1}$$

This is a concurrent set of actions, each of which carries its own monitor process CheckSit to determine if the world and reactor present an appropriate situation with respect to task a in which the action (sequence) should be carried out. A situation is a task and world-model specific invariant on the world. $sit(a)$ is the set of situations for task a for a particular world model. Every action is 'guarded' by an invariant I that monitors the executability conditions for the action; should the invariant monitor fail, that plan segment is disabled, and the planner is notified.

Control Property: Timeliness. A reaction is a response by the reactor to a situation recognized by one of the situation monitor processes in (1). We define a reaction R to be *timely* if it obeys an interval time constraint S in responding to a situation s as indicated by $T(\mathtt{S},\mathtt{R})$ in the following:

$$\begin{array}{llcl} Specification & \mathtt{S} & = & (\mathtt{Sit}_s, \mathtt{Int}_{t1,t2}; \mathtt{A}) \\ Reaction & \mathtt{R} & = & \mathtt{Checksit}_s\mathtt{@D}_c :; (\mathtt{D}_a; \mathtt{A}) \\ Timeliness & T(\mathtt{S},\mathtt{R}) & : & t1 \le c+a \le t1+t2 \end{array} \tag{2}$$

Scheduling Property: Loading. Ignoring the mechanisms by which the planner changes the reactor, the reactor can be thought of as a concurrent set of reactions in the form (1). Ultimately, these have to carried out on a computer with limited CPU resources. The CPU requirements of the reactor can be captured by the time constraints:

$$\Big|_{i\in C} (\mathtt{D}_{ti}; (\mathtt{P}_i\mathtt{@D}_{di})) :; (\mathtt{D}_{tai}; (\mathtt{A@D}_{dai})) \tag{3}$$

A situation monitor process $i \in C$ uses a CPU for periodic intervals $[ti, ti + di], [2*ti+di, 2*(ti+di)], \ldots$, assuming it doesn't trigger its consequent process. Let I_i be the set of such intervals for each process $i \in C$. Let I_i^m be the subset of each I_i chopped off after the least common multiple of the periods of all $i \in C$. We now inspect the degree of concurrency in this schedule. The tagged intersection of I^m is the set of pairs defined as follows:

$$\bigsqcap_{i\in C} I_i^m = \{(a,b) \mid a \in \bigcap_{i\in C} I_i^m,\ b = \{j \mid a \in I_j^m\}\ \} \tag{4}$$

We define Ξ_C to measure the concurrency between processes in C:

$$\Xi_C = \pi_2 \bigsqcap_{i\in C} I_i^m \tag{5}$$

(where π_2 is projection onto the second component). We say that the reactor is *overloaded* if for a CPU resource limit K we have that

$$L = \underset{s \in \Xi_C}{\mathrm{MAX}} \mid s \mid \; > K \tag{6}$$

A reactor may be underloaded but might become overloaded if some situation monitors trigger their actions. Let IA_i be I_i extended to include action intervals after *every* monitor interval (worst case scenario) and ΞA and LA be the resultant concurrency measures. An underloaded reactor for which $LA > K$ is called *fragile*.

5 Conclusion

A methodology for representing the real-time characteristics of a reactive system has been introduced. The methodology handles temporal properties associated with the reactor control problem and those associated with the CPU usage of the reactor implementation.

References

[1] R. Brooks. A robust layered control system for a mobile robot. *IEEE J. Rob. & Aut.*, RA-2(1):14–22, Mar. 1986.

[2] D.M. Lyons and M.A. Arbib. A Formal Model of Computation for Sensory-Based Robotics. *IEEE Trans. Rob. & Aut.*, 5(3):280–293, June 1989.

[3] D.M. Lyons, A.J. Hendriks, and S. Mehta. Achieving robustness by casting planning as adaptation of a reactive system. In *IEEE Int. Conf. Rob. & Aut.*, Apr. 7–12th 1991.