A Practical Approach to Integrating Reaction and Deliberation

D. M. Lyons and A. J. Hendriks
Philips Laboratories
North American Philips Corporation
Briarcliff Manor NY 10510
dml@philabs.philips.com

Abstract

A robot system operating in an environment in which there is uncertainty and change needs to integrate the ability to react with the ability to plan ahead. In this paper, we present a novel approach to this integration for a practical robot problem, the *kitting robot*. In our approach, planning is cast as the ongoing adaptation of a reactive system, incrementally bringing its behavior into line with a set of goals. A method is presented whereby the planner can make iterative improvements to the reactor so that it eventually converges on the ideal reactor (a concept related to the universal plan) for its environment and set of planner goals. To balance this theoretical work, our current implementation on a Puma 560 robot is overviewed.

1 INTRODUCTION

We are interested in building robot planning systems that operate in real-world, commercial/industrial robotic applications. This means developing systems that can cope with uncertain and changing environments, not just in theory or in simulation, but also in practice. This constraint has led us to a novel approach to integrating reaction with planned and guaranteed behavior.

We take the kitting robot (Sellers & Nof 1989) as exemplary for our domain of interest. A kitting robot is a robot system that puts together assembly kits — trays containing all the necessary parts from which to build a specific product. Simpler and cheaper automation can construct the assemblies once they have been placed in the kits and routed appropriately. So rather than having to build a factory full of expensive, intelligent robots, a manufacturer can focus the intelligence and cost into a small number of kitting robots which feed the rest of the factory. The key characteristic of the kitting robot is not so much its ability to reason about assembly, but rather its ability to choose timely and effective actions to suit an uncertain and changing environment. To deal with events as they happen, the kitting robot needs to be capable of *real-time, reactive response*. On the other hand, to deal with changes in the environment (e.g., bursts of errors in parts, or failures of resources) or new goals issued by factory management, the kitting robot needs to be able *to plan ahead 'on the fly'*. Thus we need to integrate the versatility and robustness of the reactive systems described by Brooks (Brooks 1986) and others, with the *a-priori* deliberation of traditional planning systems. In addition, to be successful in a real-world commercial/industrial application, the kitting robot, like any other piece of machinery (or indeed any employee) needs to be capable of operating reliably within a set of specifications. Therefore, our objective is to build a system with a sufficiently formal basis that it is possible to give engineering guarantees about its behavior before it is put into operation. Our focus in this paper will be on the issue of integration of reaction and deliberation for such systems; in particular in the case of mostly repetitive behavior such as robotic kitting.

In the next section, we look at some of the existing work on the problem of integrating reaction and deliberation from our perspective of the kitting robot problem. Section 3 presents an overview of our approach to this problem: we cast planning as the *on-going, incremental* adaptation of a separate and concurrent reactive system. Our need for guaranteed behavior leads to the following constraint: we insist that the planner only make adaptations to the reactive system when these adaptations are improvements. Section 4 presents the detailed description of how we represent reactors and reactor adaptation (Lyons et al 1991). In section 5, we discuss our approach to generating adaptations to meet this constraint. Central to this section is our definition of a *situation*: a unit with which to decompose problem spaces in a manner appropriate to reasoning about reactions. Section 5 overviews our testbed implementation.

2 INTEGRATING REACTION AND DELIBERATION

The classical definition of a planner is a program that generates a sequence of operations, the plan, which when applied, will take the world from its initial state to a goal state. A plan is first generated and then sent to an executor which carries out each of the actions. This approach becomes very fragile in a dynamic and changing environment (McDermott 1991, Lyons & Hendriks 1992). The concept of reactive systems, e.g., (Brooks 1986), (Agre & Chapman 1987), was introduced to address this problem. Such a system doesn't plan in the classical sense, but rather reacts directly to its environment. These systems are surprisingly robust and versatile, however, they do not support *a-priori* deliberation about action sequences. Our kitting robot application demands both reactive abilities and *a-priori* deliberation. This integration places constraints upon both the architecture of the planning and reacting system and also the plan representation.

2.1 ARCHITECTURE

Schoppers introduced the notion of *universal plans* to combat the fragility of classical planning in realistic environments (Schoppers 1989). A universal plan guarantees the eventual achievement of its goals, no matter what the initial state of the environment (cf. STRIPS triangle tables). Such plans are essential when a robot cannot accurately predict the current state of the environment. Unfortunately, for most realistic domains, universal plans tend to be very large and extremely complex. While constructing a universal plan may be an appropriate long-term objective for a planner, in practice we will need to settle for a short-term approximation to it.

In the field of robotics, the typical approach to integrating planning and reaction has been to 'piggy-back' a planner on top of a reactive scheduler/executor, e.g., SROMA (Xiadong & Beckey 1988) for robotic assembly, and the trajectory planner (Fraichard & Laugier 1991) for robot motion planning. An abstract 'plan' is generated *a-priori* and then fed to a reactive scheduler or executor which is allowed to modify action ordering or introduce minor deviations. We argue that the resultant system behavior is only partially reactive for the following reasons:

1. the generation and loading of entire plans is simply not feasible for most uncertain and changing environments, and
2. such a reactive scheduler/executor can only 'react' while it has a plan; a true reactive system needs to be able to act at *any* time.

Bresina & Drummond (Bresina & Drummond 1990, Drummond & Bresina 1990) describe an architecture, ERE, that integrates planning, scheduling and control. Their architecture consists of a *reactor*, that immediately responds to the environment, a *projector*, that expands possible futures and advises the reactor on appropriate ones, and a *reductor*, that provides search control to the projector. Bresina & Drummond emphasize the *anytime* property of their architecture. However, ultimately this property results in their system taking any arbitrary actions whose preconditions are enabled by the state of the environment. While this approach supports an elegant inductive description of reactor refinement, in an industrial/commercial application such a strategy can be disastrous. We need to emphasize timely, relevant actions over anytime arbitrary actions.

It is also interesting that the idea of planning by adapting a reactive system has some similarity to recent machine learning architectures, e.g., DYNA (Sutton 1990) . In such systems, the reactor is constructed by the trial and error learning of the regularities in the environment. Sutton demonstrated for a simple example task that such architectures can support a mutually enriching integration of learning, planning and reaction. For the present, we will not address learning issues in our work. Instead, we will here assume that although the environment is uncertain and changing, a correct and static description of that uncertainty and change is known.

2.2 PLAN REPRESENTATION

Controlling a real robot in a realistic environment puts great demands on the plan representation: the borderline between robot plan representation and robot programming language begins to blur (McDermott 1991). Conditional and iterative commands are necessary in the plan, as are commands to access and process sensor data and to set actuator signals. McDermott suggests an improved version of Firby's RAPS (Firby 1987) as an appropriately detailed plan/programming representation. A plan representation does have one additional requirement above those of a robot programming language: it must be possible to predict and reason about the behavior that the plan causes the robot to exhibit. It is not yet clear that RAPS meets this requirement.

Lyons & Arbib (Lyons & Arbib 1989) developed a special model of computation for sensory-based robot programming. That work was begun by putting together a list of the computational characteristics of the robot programming domain. The principle among these were:

1. robot programming is inherently parallel and based on building hierarchical connections between sensors and actuators, and

2. the principle objective of a robot program is to have some effect on the external world.

The model developed was called $\mathcal{RS}$ (Robot Schemas), and had the following style: All actions, whether computational or physical, were cast as concurrent processes, and programs/plans were cast as networks of these processes. That paper provided a port automaton formalism for reasoning about the behavior of these networks. In (Lyons 1989, 1990) we extended the representation and formal analysis vocabulary of $\mathcal{RS}$ using the techniques of the process algebra field (Hoare 1985): We investigated representing and reasoning about interacting plans and world models built by composing processes together using a set of formally defined process composition operators. This process style has much in common with Nilsson's ASTRAL language (Nilsson 1988). A crucial point of similarity is that both models address the issue of 'communications' between actions *while* the actions are in progress. This capability is important in plans that interact with an uncertain world.

3 PLANNING AS ADAPTATION OF A REACTIVE SYSTEM

Our definition of planning is akin to that of Bresina & Drummond, and more recently, of McDermott (McDermott 1991): a planner is a system that continually modifies a concurrent and separate reactive system so that its behavior becomes more goal directed. We will refer to the reactive system as the *reactor* after the terminology established by Bresina & Drummond. Figure 1 illustrates our architecture. A reactor, a network of sensory-motor reactions, interacts directly with an uncertain and changing environment. The planner is a concurrent and separate system that monitors the reactor and its environment by issuing perception requests. These are distinct from the reactor's sensory actions. When, in the planner's view, the reactor structure will not produce behavior that conforms to the planner's goals, the planner issues adaptations, structural changes to the reactor, to bring reactor behavior back into line with the goals. Our focus in this paper will be on the interaction between planner and reactor; in particular, on the cumulative effect of planner modifications and on the necessity for the planner to be able to introduce perception actions potentially separate and distinct from reactor sensory actions.

Figure 1: Planner-Reactor-World System

3.1 REACTOR

The reactor contains a network of sensory-motor 'linkages' called reactions. In our framework, these reactions are sensory processes composed with effector (or action) processes in such a fashion that should the sensory process detect its triggering condition, then it will in turn trigger the effector process. The reactor has several key properties:

1. It can produce action at *any* time. Unlike a plan executor, a reactor can act independently of the planner; it is always actively inspecting the world, and will act should one of its reactions be triggered. A reactor should produce useful behavior even without a planner.

2. It is a *real-time* system. The central objective of the reactor is to produce the appropriate action at the appropriate time. To make this possible, all the 'knowledge' in a reactor is encoded into reactions and cross-connections between them, and annotated with time-constraints.

The reactor encapsulates what would typically be called the plan in more traditional approaches. By making it a completely separate system, we remove the delays involved when a general deliberation subsystem is always interposed between sensing and action. The reactor is also responsible for the default behavior of the system in the following sense: It is initialized with a number of generally useful behaviors already in place, for example, distinguishing objects from background, locally avoiding obstacles, and scanning its sensors around the environment. Note that our reactor differs from that proposed by Bresina & Drummond in that it does not, by default, contain all the action descriptions and their preconditions. The inclusion of these reactions is what leads to their reactor producing arbitrary enabled actions on occasion.

3.2 PLANNER

The planner is completely separate from, and concurrent with, the reactor. Rather than seeing the planner as a higher-level system that loads plans into an executor, we see the planner as a system that continually refines the reactor to produce appropriate behavior (Figure 1). The input to the planner includes: a model of the environment (EM) in which the planner is operating; a description of the reactor (R); and advice (G) from a user about objectives the reactor should achieve (e.g., in the kitting problem domain, geometric goals such as kit layouts) and constraints the reactor should obey in its behavior (e.g., batch mix or resource usage constraints).

The planner continually determines from R if the reactor's behavior in the current environment will indeed conform to the restrictions G. If the planner finds that the reactor will not obey the restrictions, it makes an incremental change to the reactor configuration to bring the reactor more into line with G. As indicated in Figure 1, there are two routes of interaction between planner and reactor.

1. Adaptations. The planner alters the structure of the reactor by specifying adaptations.
2. Perceptions. These are sensory data collected by the reactor to be sent to the planner.

An adaptation is essentially an instruction to delete part of the reactor structure or to add in some extra structure. The mechanism of adaptation is described in detail in the next section. Each individual adaptation should be small in effect and scope; large changes in reactor behavior only come about as the result of iterated adaptation. Large adaptations would be equivalent to downloading 'plans' to the reactor; something we need to avoid.

The knowledge needs of the planner and reactor are almost always different. The reactor uses sensory data to determine whether to fire its reactions. The planner needs sensory data to allow it to predict the future state of the environment. Thus, in addition to being able to adapt the reactor to contain new reactions, the planner needs to be able to insert its own sensory data collection processes. To distinguish between reactor and planner sensory data, we refer to any data collected by the reactor as sensory data, but sensory data collected to be sent to the planner will be referred to as perceptual data. Without any sensory data, the planner is detached from the environment. In contrast, systems with share a sensory data-base between the reactive and deliberative components force the latter to receive all and only all the sensory data that the reactive component accumulates. As a result, the deliberative component ends up with too much uninteresting data and, more seriously, missing data needed only by itself. The perceptual data needed will be strongly dependent on current goals, and will most likely change as those goals change.

4 MECHANISM OF ADAPTATION

The section reviews our progress in representing the reactor using the $\mathcal{RS}$ model. We begin with a short review of $\mathcal{RS}$ notation. For a more detailed motivation and description see (Lyons 1989,1990) .

4.1 $\mathcal{RS}$ NOTATION

Plans, actions and world models are all represented in $\mathcal{RS}$ as hierarchical networks of concurrent processes. A description of a process, or network of processes, is called a *schema*. For example, $\mathtt{Joint}_v\langle x\rangle$ denotes a process that is an instance of the schema `Joint` with one ingoing parameter v and one outcoming result x. Networks are built by composing processes together using several kinds of *process composition operators*. This allows processes to be ordered in various ways, including concurrent, conditional and iterative orderings. At the bottom of this hierarchy, every network must be composed from a set of atomic, pre-defined processes. The set of *basic schemas* defines what processes are atomic. The main composition operations are *sequential* `A;B`, *concurrent* `(A,B)`, *conditional* `A:B` and *disabling* `A#B`.

Sequential composition is used to enforce a strict ordering on operations. For example, ensuring that $part2$ is always placed after $part1$: $\mathtt{Place}_{part1};\mathtt{Place}_{part2}$.

Concurrent composition indicates that two or more processes should be carried out concurrently. This allows us to represent a lack of ordering between activities, e.g., $(\mathtt{Place}_{part1}, \mathtt{Place}_{part2})$, or parallel actions — actions which need to be done simultaneously, e.g., squeezing an object obj with two fingers $f1$ and $f2$:

$(\mathtt{ApplyForce}_{f1,obj}, \mathtt{ApplyForce}_{f2,obj})$. A set of concurrent processes $\mathtt{A}^i, i \in P$ is written $\Big|_{P} \mathtt{A}^i$.

Conditional composition allows the construction of networks whose behavior is conditional. The network of $\mathtt{T} = \mathtt{P:Q}$ behaves like `P;Q` iff `P` terminates successfully. If `P` aborts, then `Q` is not carried out, and `T` aborts. For example, in $\mathtt{LidOpen}_{box} : \mathtt{Place}_{x,box}$ whether `Place` is carried out or not depends on whether `LidOpen` terminates successfully or not.

Disabling composition allows one process to terminate another. The network `T = A#B` behaves like `(A,B)` except that it terminates whenever either process terminates.

Finally, two non-atomic composition operations are defined in terms of these four: *synchronous recurrent* $\mathtt{A} :; \mathtt{B} = \mathtt{A} : (\mathtt{B}; \mathtt{A} :; \mathtt{B})$, and *asynchronous recurrent* $\mathtt{A} :: \mathtt{B} = \mathtt{A} : (\mathtt{B}, \mathtt{A} :: \mathtt{B})$. Synchronous recurrent composition is similar to *while-loop* iteration. Asynchronous recurrent composition does not iterate, but rather 'spawns' off a set of concurrent processes every time its 'condition' is satisfied.

It is important to represent how process networks evolve over time, as processes dynamically terminate or are created. This is captured by the evolves operator: We say that process `A` evolves into process `B` under condition Ω if `A` becomes equal to `B` when condition Ω occurs; we write this as $\mathtt{A} \xrightarrow{\Omega} \mathtt{B}$. Evolves is axiomatized from the definitions of the composition operators, e.g., $\mathtt{A}; \mathtt{B} \xrightarrow{\Omega} \mathtt{B}$.

4.2 REACTOR ADAPTATION

The reactor consists of a set of guarded sensory-action pairs (reactions) and a well-defined interface through which the planner can modify the reactor structure:

$$\mathtt{REACTOR} = (\quad \Big| \; \mathtt{E}^i \# \mathtt{P}^i \;, \; \mathtt{ADDPLAN}\langle k \rangle :: (\mathtt{E}^k \# \mathtt{P}^k) \;) \quad (1)$$

where

1. $\mathtt{P}^i$ is a reaction or reactor segment (as explained later).
2. $\mathtt{E}^i$ is a guard process by which (by the #-composition) $\mathtt{P}^i$ can be removed by the planner.
3. `ADDPLAN` is the interface through which the planner can add (by the ::-composition) new guarded reactions $\mathtt{E}^m \# \mathtt{P}^m$.

Reactions — hard-wired responses to sensory conditions — are specified in $\mathcal{RS}$ as compositions of sensory and motor processes. For example, a useful reaction for a kitting robot might be to notice anytime some instance of part $p1$ arrives in its buffer area and to acquire that object instance and move it into the workspace. This can be expressed as follows:

$$\mathtt{Locate}_{p1}\langle obj \rangle :; \mathtt{Place}_{obj,dest}$$

where $\mathtt{Locate}_m$ is a basic sensory process that inspects the world for an instance of part m and terminates successfully when it finds one, producing a pointer to the instance in obj. `Place` then acquires and moves object instance obj to location $dest$. The recurrent composition ':;' forces the network to be *continually* recreated.

A reactor does not contain/execute plans in the same sense that a classical plan executor/scheduler does, i.e., as a data structure that is manipulated or executed. Nonetheless, certain parts of the reactor structure can be identified as corresponding to the instructions necessary for a specific task or subtask; we call these parts *reactor segments.* They are represented in a *situation-labeled* form:

$$\mathtt{P}^t = \Big|_{s \in sit(t)} \mathtt{CheckSit}_s : \mathtt{Reaction}_{t,s} \# I_{t,s} \qquad (2)$$

as a concurrent set of reactions, each of which carries its own monitor process `CheckSit` to determine if the world and reactor present an appropriate situation with respect to task t for that reaction to be active. $sit(t)$ is the set of situations for task t for a particular world model. We postpone to the next section a more detailed description of our situation concept and how to get $sit(t)$. Every segment is 'guarded' by a condition monitor `I` that monitors the applicability conditions for the segment; should the monitor fail (`I` terminates), that segment is disabled (by the #-composition), and the planner is notified. Again, we postpone justifying this construct until the next section.

We can write down how the reactor is affected by the changes described in items 2 and 3 of eq. (1):

$$\begin{array}{ccc} (\mathtt{R}, \mathtt{E}^k \# \mathtt{P}^k) & \xrightarrow{Dk} & \mathtt{R} \\ \mathtt{R} & \xrightarrow{Ak} & (\mathtt{R}, \mathtt{E}^m \# \mathtt{P}^m) \end{array} \qquad (3)$$

$$\textit{where } Dk \textit{ is } \mathtt{E}^k \longrightarrow \mathtt{STOP}$$
$$\textit{where } Ak \textit{ is } \mathtt{ADDPLAN}\langle k \rangle \longrightarrow \mathtt{STOP} \;\&\; k = m$$

That is Dk can be used to *delete* reactions, and Ak can be used to *add* reactions. Thus, we can express the process of reactor adaptation as follows

$$\mathtt{R} \xrightarrow{\alpha} \mathtt{R}' \qquad (4)$$

where `R'` is the adapted reactor and α is some combination of Ak and Dk conditions. This adaptation mechanism describes the way in which the planner can modify the reactor to improve its behavior.

5 INCREMENTAL REACTOR REFINEMENT

In our framework, planning is a continual process of accepting goals from a user and perceptions from the reactor, and producing incremental adaptations to the reactor, when necessary. Since our problem domain requires carrying out repetitive tasks within an uncertain and possibly changing environment, the speedy construction of an initial working reactor as well as the incremental generation of reactor improvements are important.

This section first describes the ultimate objective of the planner, the *ideal reactor*. However, since it is very unlikely that a planner can generate this ideal reactor at once in a timely fashion, we describe our approach to initially restricting the planner's objective (by restricting the domain) so that it can produce an initial reactor more quickly. The subsequent incremental lifting of these restrictions then allows the planner to incrementally construct reactor improvements, while *at all times* maintaining a working reactor structure. Lastly, we examine in more detail the way the planner arrives at a reactor adaptation from a set of goals G, an environment model EM and a reactor model R_t.

5.1 THE IDEAL REACTOR

The ultimate objective of the planner is the construction of a reactor that achieves the goals G not only in the current environment, but because change is possible, in all environments that might occur, as indicated by the environment model. A reactor that fulfills this criterion is called an *ideal reactor*, and is similar in behavior to Schoppers universal plans.

For a given goal G and environment model EM, let the ideal reactor be modeled by the $\mathcal{RS}$ process $\mathtt{R}^*$. Then, the behavior of this ideal reactor, operating in the environment, modeled by the process $\mathtt{EM}$, is described by their *concurrent composition*, written as $(\mathtt{R}^*, \mathtt{EM})$. To analyze how their joint behavior evolves over time, we use the $\mathcal{RS}$ evolution operator. The behavior of the ideal reactor $\mathtt{R}^*$ operating in environment $\mathtt{EM}$ is characterized by the following evolution equation:

$$(\mathtt{R}^* \ , \ \mathtt{EM} \ ) \longrightarrow (\ \mathtt{R}^* \ , \ \mathtt{GEM} \ ) \tag{5}$$

That is, when the ideal reactor is run concurrently with a environment model process, the latter will always evolve to an environment in which the goal is satisfied $\mathtt{GEM}$.

By this definition, the ideal reactor is the most robust reactor possible; however, it is unrealistic to suppose that a planner will always be able to generate the ideal reactor in one step for the following reasons:

1. it may not have sufficient time because of the time constraints within the reactor;
2. there may be too much uncertainty in the world model for the reactor to be constructed without doing some more sensing; or,
3. it may not be possible to build the ideal reactor because of resource and action constraints.

Thus, while the ideal reactor is an eventual target for the planner, in general it cannot be constructed all at once. It becomes important therefore to find a method that constructs the ideal reactor in 'useful' increments.

5.2 INCREMENTAL CONSTRUCTION OF THE IDEAL REACTOR

A natural constraint to make is that any change to the reactor should result in another working reactor which is capable of behavior at least as goalworthy as the the original reactor. To investigate this issue, we can restrict the ideal reactor definition above (5) with conditions:

$$(\ \mathtt{R}, \ \mathtt{EM} \ ) \xrightarrow{\omega} (\ \mathtt{R}, \ \mathtt{GEM} \ ) \tag{6}$$

Thus, as long as conditions ω hold, $\mathtt{R}$ behaves like the ideal reactor. We call this the ω-ideal reactor $\mathtt{R}^\omega$ for environment EM and goals G.

In planner terms, ω is a set of assumptions under which $\mathtt{R}$ has been constructed. This suggests the following incremental plan construction strategy: ω is initially chosen to allow the planner to quickly produce a working reactor and then ω is gradually relaxed over time. We can rewrite the evolution of the reactor as:

$$\begin{array}{c} \mathtt{R}^{\omega 1} \xrightarrow{\alpha 1} \mathtt{R}^{\omega 2} \xrightarrow{\alpha 2} \mathtt{R}^{\omega 3} \xrightarrow{\alpha 3} \ldots \xrightarrow{\alpha n} \mathtt{R}^* \\ for \ \ \omega 1 > \omega 2 > \omega 3 > \ldots > \emptyset \end{array} \tag{7}$$

where the αi are the adaptation commands generated by the planner.

The assumption relaxation priorities are governed by the chosen ordering on ω. Using the likelihood of ω holding in the environment as the ordering criterion enables the planner to build reactor adaptations to abolish the least likely assumptions first. It will take care of increasingly more likely assumptions as it has time. Of course, at any point the reactor may be forced to deal with the fact that it's assumptions *do not hold* in the current environment. As it stands, eq.

(6) claims nothing about the behavior of the reactor if the assumptions don't hold. In the case of failed assumptions, it is necessary that the reactor do nothing that would prohibit a future improved version of the reactor from operating successfully. Sometimes it is okay to simply disable the inappropriate behavior. But in other cases, it may be necessary to actually carry out some actions to prevent harmful side effects from the failed assumption. As of yet, we have not developed a general approach to this part of the problem; we describe our partial solution below.

To guard against failing assumptions, it is important that the ω-ideal reactor contain assumption condition monitors that 'protect' ω. For example, a reasonable assumption to make in the kitting robot problem is that the parts to be kitted will not move (on their own) once they have been delivered to the workcell. The planner can built a reactor $\mathtt{R}^{static}$ based on this assumption and over time would eventually adapt this reactor to deal with moving objects. In the interim, $\mathtt{R}^{static}$ needs to include monitors that detect the violation of the static position assumption, i.e., that trigger when part positions change inexplicably. The triggering of an assumption monitor should force the planner to prematurely relax the violated assumption.

5.3 INCREMENTAL CONSTRUCTION OF THE ω-IDEAL REACTOR

For the same reasons that the ideal reactor can't be produced in one increment, it is unrealistic to believe that the ω-ideal reactor can be produced in one increment. Thus the planner needs to be able to incrementally adapt a reactor $\mathtt{R}$ to become $\mathtt{R}^{\omega}$.

Any reactor $\mathtt{R}$ can be seen as a part of the ideal reactor, since we can always choose a set of reactions that inhibit reactions in $\mathtt{R}$ not in $\mathtt{R}^{\omega}$ and add in reactions in $\mathtt{R}^{\omega}$ missing from $\mathtt{R}$. At every planner stage t, based on the environment model EM_t and current goals G_t, the planner generates what we call an *expectation*, E_t, an abstract description of the changes it expects to have to make to the reactor R_t to achieve G_t. It expresses the planner's confidence that it can achieve its goals. At every stage, therefore, the combination of the reactor model R_t and the expectation should be exactly the ω-ideal reactor, constraining the reactor adaptation ΔR_t to be an improvement (where ω is the current set of assumptions the planner is working with):

$$R_t + E_t^{\omega} = R_t^{\omega} = (R_t + \Delta R_t) + (E_t^{\omega} - \Delta E_t^{\omega}) \qquad (8)$$

The process of incremental improvement of the reactor can now be seen as incremental reduction of the planner's expectation by some ΔE_t and corresponding improvement of the reactor by some corresponding ΔR_t, until when E_t is null, the current reactor is exactly the ideal reactor.

Figure 2: Example Situations.

To reduce the expectation, the planner reasons within a problem solving context consisting of the relevant parts of the Environment Model, the action repertoire and the assumptions ω. The outcome is a reactor adaptation ΔR_t, the reactions necessary to implement the expectation reduction ΔE_t. The manageable size of ΔE_t is constrained by the following factors:

1. The response time of the planner. This can come from the design specification of the planner, or from the temporal constraints on goals and actions in the plan.

2. The effects of uncertain knowledge on the plan; the planner may need to issue perception requests before more of the expectation can be reduced.

3. Resource use and mappings.

The planner maintains its expectation as the time-ordered set of *situations* that it needs to address in more detail. Formally, situations are equivalence relations on the state-space for a specific task in a specific environment. Multiple situations may be active at one time. For the reactor, a *situation* being *active* expresses the appropriateness of a particular set of reactions; those *associated with* that situation.

5.4 SITUATIONS

To illustrate the concept of a situation, take for example, a kitting robot assembling a tray with three independent parts a, b and c. There are multiple orders in which the parts can be placed into the tray, all of which lead to the situation in which the tray is full, and all of which are equivalent (Figure 2(B)). At one level of abstraction, the kitting task can be divided into three sequential situations: *notray*, in which the reactions that locate and transfer trays into the workspace are active; *tray,* in which the reactions to fill a tray are active; and, *fulltray,* in which the reactions to transfer a tray out of the workcell are active (Figure 2(A)). Situations can be nested hierarchically, though this nesting is not as straightforward as for states. The *fill-kit* situation can be decomposed into three concurrent situations (Figure 2(C)) *lacks-a, lacks-b* and *lacks-c*; where *lacks-a* means that the kit is still missing part a, and it enables the reactions which find and place part a. Any combination of these three situations may be active within *tray,* and that combination can change as parts are successfully placed.

Situations have the following useful characteristics:

1. They lend themselves to the use of reactions: Within a situation, the planner doesn't really care what path the reactor takes to achieve the goal. Thus, the actions in a situation can be handled by a set of reactions keyed to fire in that situation.

2. They facilitate incremental construction: The situation defines a unit of useful activity, so the planner can reduce its expectation in units of situations.

3. They facilitate relaxation of assumptions: The relaxation of an assumption will always result in either addition of one or more new situations, (reflecting the extra work needed in the absence of the assumption), or the addition of additional reflexes into a situation (if removal of the assumption can be covered by a generalized behavior within the same situation), or both.

5.5 PERCEPTION

In our framework, we have made a clear distinction between perception for planner purposes and sensing for reactor purposes. In addition to being able to adapt the reactor to contain new reactions, the planner can also adapt the reactor to contain its own sensory data collection processes.

The use of perception is a key part of the planning process, allowing a planner to "familiarize" itself with the environment, resolve uncertainties and prioritize its work by the state of affairs in the environment. We distinguish three types of perception:

1. Direct perceptions — these resolve uncertainties.

2. Situation monitors — these signal the occurrence of situations.

3. Assumption monitors — these signal the failure of an assumption.

Direct perceptions are perception processes that the planner inserts in the reactor to resolve uncertainties in its model of the environment. The other two types help the planner to focus and prioritize its remaining work to adapt the reactor. Situation monitors are perception processes inserted in the reactor to signal the planner that a situation has become active, in the case that that situation is still part of the planners expectation and thus needs to be worked out in detail. Finally, assumption monitors are perception processes inserted in the reactor to warn the planner that a particular assumption has failed and needs to be relaxed for the reactor to behave adequately again.

Thus through the use of perception, the planner can always work on relevant parts of problem first, whether that be reducing expectations to concrete reactor adaptations or relaxing assumptions to achieve more robust behavior.

6 TESTBED ENVIRONMENT

We have implemented a planner-reactor system for kitting robot experiments (figure 3).

6.1 HARDWARE

A Puma-560 robot is the basis of the kitting workcell. The original VAL-based controller is used to control the manipulator. The Puma carries a four-fingered, 2-DOF pneumatic gripper, equipped with finger force and position sensors. The workcell has two cameras: A global camera situated above the workcell, whose view covers the entire workspace, but consequentially doesn't yield much detail; and a local camera, embedded in the 'palm' of the gripper, whose view is determined by the position and pose of the gripper, and which can be used to get a close-up view of an object.

6.2 REACTOR

A subset of the $\mathcal{RS}$ model has been implemented as a robot programming language. The `RS-L3` programming environment consists of a YACC/LEX-based parser and a real-time executor. The reactor is a set of `RS-L3` process definition equations, as described in Section 4, which allow for incremental additional and deletion of reactor structure.

Figure 3: The Kitting Robot Implementation.

6.3 PLANNER

The planner uses Pelavin's Interval Temporal Logic (ITL) formalism to represent world models (Pelavin 1988). It maintains these models as sets of temporal predicates and rules. Temporal relations between intervals of time are maintained in a network and new relations are enforced by constraint propagation (Allen 1983). Actions have executability condition and effect formulas. Once the executability conditions are satisfied, execution of the action will guarantee that the effects will occur in the world. The use of ITL as the basis for our planner enables us to specify and reason about communicating concurrent actions, since executability conditions can include conditions that must hold *while* the action is executing. The planner maintains the link with the reactor using asynchronous signal handling to catch incoming perceptions.

For more details of this testbed, see (Lyons & Hendriks 1991).

7 DISCUSSION

7.1 SUMMARY

This paper built upon our reactor description in (Lyons et al 1991) and described the objectives and internal structure of the planner component. In particular, it took a closer look at the cumulative effects of planner adaptation actions. To support iterative improvement of the reactor, we introduced the concepts of planner *expectation*, the production of *perception processes* by the planner, and the use of *situations* as a modular unit. Our current implementation environment was briefly described.

7.2 COMPARISONS

Our approach is centered on the concept of having separate, but coupled, planner and reactor systems. It is possible to combine planning and action in a single architecture, e.g., Robo-Soar (Laird 1990). While this has tremendous advantage in allowing for arbitrary interaction between planning and execution, it also has the disadvantage that it allows for arbitrary interference between execution and planning. For example, when an event that should be reacted to occurs, Robo-Soar loses its intermediate planning state in order to react to the event.

The most directly related work to ours is that of Bresina & Drummond . The principle differences with their work are: The $\mathcal{RS}$ model gives us a formal framework to explore the problems of iterative adaptation. Our planner and reactor are less tightly coupled than theirs (discrete adaptations and perceptions are the only channels of interaction), but *mutually* influence each other. Finally, we use a planner based on the concept of situations and reducing expectation to ensure that the reactor is always improved by planner adaptations.

7.3 FURTHER EFFORT

We believe that it is possible to construct a stronger theoretical treatment of the iterative effects of planner adaptation. In basic design, the planner-reactor falls into the class of what in adaptive control are called *self-tuning regulators*. $\mathcal{RS}$ provides us with a powerful discrete-event control representation: more concise than automaton or petri-net models, and yet with some mathematical tools (such as the evolves operator) available. A stronger theory will also help the issues of what constitutes a good base-line reactor and the observability of assumptions.

References