Achieving Robustness by Casting Planning as Adaptation of a Reactive System.

D. M. Lyons, A. J. Hendriks, S. Mehta

Philips Laboratories
North American Philips Corporation
Briarcliff Manor NY 10510

Abstract

Classical AI planning is not sufficiently robust in uncertain and dynamic environments. Reactive approaches are robust in some environments – namely those for which they have been pre-programmed. An approach is presented here that integrates *a priori* planning with reaction. As motivation, a practical robot problem, the kitting robot, is presented. Solving this problem demands a system that can make timely and robust actions in an uncertain environment. A solution to the kitting robot problem is outlined in which planning is cast as adaptation of a reactive system to suit changes in the goals or environment. The reactive system (the *reactor*) is based on a formal model for representing flexible robot plans, the $\mathcal{RS}$ model. Thus, we were able to formalize the mechanisms by which the planner improves the behavior of the reactor. This system was implemented to control a Puma-560 robot equipped with visual sensing.

1 Introduction

Solving the problem of planning and acting robustly in an uncertain environment is a crucial objective in robotics. For example, Fox & Kempf [7] describe how scheduling manufacturing operations suffers from the problem of executional uncertainty: During the execution of a schedule, some of the resources can be expected to become unavailable, but there is no way to tell in advance which one or when. We introduce a practical problem, the kitting robot problem [19], which necessitates producing a robot system that can plan and react in an uncertain environment, and we describe an integrated approach to planning and reacting to address this kind of problem.

Classical AI planning, as exemplified by STRIPS [6] and upon which many planning systems are still based, is not sufficiently robust in uncertain and dynamic environments[17]. Reactive approaches, a more recent development exemplified by Brooks' work[4], operate very robustly in some uncertain and dynamic environments — namely those for which the reactions have been pre-programmed. Our solution to the kitting robot problem improves on the state-of-the-art by combining the ability to plan in advance with the ability to react at *all* times to the environment. We cast planning as *adaptation* of a reactive system (the Reactor) to suit changes in goals or the environment. Rather than generating and then executing plans, our approach is to have the Reactor be a real-time system that continually interacts with the environment, and the planner be a separate and concurrent system that 'tunes' the behavior of the reactor to ensure that goals are achieved. The Reactor is based on a formal framework for representing flexible robot plans, the $\mathcal{RS}$ model[12, 14]. Thus, the behavior of the Reactor, and the rules by which the Reactor can be modified, become open to mathematical analysis. We present a set of rules by which the planner 'improves' the reactor to render its behavior more goal-directed. Adaptation of the reactor by the planner is made possible by the unique way in which we represent actions as processes that can communicate with, and be affected by, other actions. The reactor is implemented on a real-time kernel that directly emulates the $\mathcal{RS}$ model [10, 15]. The planner is implemented under the POPLOG system on a SUN3. The planner-reactor system is used to control a Puma-560 robot equipped with two cameras and a force-sensing, multi-fingered gripper.

Section 2 introduces the kitting problem and describes why the current state of the literature does not support solving it. Section 3 introduces the 'planning as adaptation' concept in more detail, and some background information on our action representation, the $\mathcal{RS}$ model. Section 4 describes each component of the system in more detail, and finishes with an example of the system in operation.

2 Kitting Robot

A kitting robot is a robot system that puts together assembly kits. Simpler and cheaper automation can construct the assemblies once they have been placed in the kits and routed appropriately. Instead of building a factory full of expensive, intelligent robots, the intelligence and cost is focused into a small number of kitting robots which feed the rest of the factory. Kitting provides the line stock availability, the line scheduling flexibility, and the reduced station cycle time productivity for competitive results [19]. However, to achieve this ideal, the kitting robot has to 'iron out' all the uncertainties associated with the assembly process so that it can be done by simpler automation.

The construction of kits is actually a simple form of assembly: there is much less sequentiality (and therefore, more freedom) in placing parts into kits than in assembly, and mating between part and kit is usually not a high-tolerance procedure. The kitting robot has to deal with the following sources of uncertainty:

1. Variable arrival rates and poses of incoming parts. The primary job of the kitting robot is to take parts (perhaps directly from stores) with high position and pose uncertainty, and to put them in a kit so as to reduce the uncertainty.
2. Errors in incoming parts. Faulty parts need to be removed before they either cause errors in downstream automation, or become assembled into (faulty or low-quality) products.
3. Substitutability of incoming parts. That is, which parts can be substituted for other, unavailable parts.
4. Mixed batches of kits. The robot may be serving more than one line, or the lines may be running mixed batches.
5. Variable availability of resources. Examples of resources are tools or machines with which the the robot needs to coordinate directly.
6. Response to events in the downstream automation. For example, machines breaking down, or alternate machine configurations.
7. Response to factory management advice. For example, response to changing the batch mix or throughput, or implementing once-off fixes for special situations.

Thus, 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 dynamic environment.

In order to deal with events as they happen, the kitting robot needs to be capable of *real-time, reactive response*. On the other hand, in order 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'*. The current literature in Robotics and AI does not yet support building a robot control system that successfully incorporates both of these attributes.

Classical AI planners, e.g., STRIPS[6], take one 'snapshot' of the world, and produce a fixed sequence of actions to achieve a goal in that world. This approach is too inflexible to work in an uncertain and dynamic environment: by the time the plan has been produced, the world may well have changed in such a fashion as to render the plan useless. There is an even more important limitation with this approach: STRIPS-like, state-based approaches make it almost impossible to represent *concurrent actions*, and the actions of *external entities* (entities other than the active agent, such as other machines, or even the forces of nature)[17]. Despite this, such planners are still common [?].

More recently, the concept of systems that react, rather than plan, has been introduced by Brooks [4] and Agre & Chapman [1] among others. This approach allows for fast and flexible response to an environment that can be complex, dynamic and uncertain. One example of the advantage of such a reactive system in an industrial setting is contained in Fox & Kempf's paper[7] on opportunistic scheduling.

There are two basic problems with the purely reactive approach: Firstly, to date the building of reactive systems has been largely manual and *ad-hoc*. We argue here that this is inappropriate. A reactive system can be expected to exhibit complex behavior in a complex environment[4]; that is one of the key advantages of the approach. However, human intuition is notoriously unreliable in predicting such complex situations. We should make every effort to use mathematical techniques for predicting and designing reactive behavior if at all possible. This was our motivation in previous work [12, 16]. Bestavros et al.[2] also describe a formal framework for building reactive systems, with many similarities to our approach.

The second problem is that a reactive approach doesn't work well for situations outside the range for which the 'reactions' were programmed[18]. There have been a number of attempts to combine the ability to react with that of *a priori* planning (usually in the context of an assembly workcell) [5, 21]. In general, these systems work by generating a complete plan in advance. This plan is then down-loaded to an Executor or Scheduler, which then chooses an appropriate action sequence based on the state of the environment. This misses an important point however. If a reactive system is indeed called for in the problem specification, then the standard planner-executor division will simply not suffice for two reasons:

1. Generation and loading of entire plans is unrealistic in a dynamic and uncertain environment.
2. An executor (or scheduler) can only act when it has been loaded with a plan; a reactive system needs to be able to act at any time.

In the remainder of this paper, we will describe a novel approach to integrating planning and reaction in the kitting robot domain. The approach has the following characteristics.

1. It is based on a precise framework for reactive plans, the $\mathcal{RS}$ model, and formal techniques for looking at the behavior of plans situated in their environment.
2. It defines planning not as the off-line *generation* of plans, but as the on-line *adaptation* of a reactive system to suit the environment and system goals.

3 Planning and Reacting

The usual definition of a planner is a program that produces a sequence of operations that when applied to the world, will take the world from its initial state to a goal state. Typically, this plan is then sent to an executer which carries out each of the actions. Our definition of planning is more akin to those of Agre & Chapman [1] or Bresina &

Figure 1: Planner-Reactor-World System

Drummond[3]: a planner is a system that continually modifies a reactive system, the reactor[1], so that its behavior becomes more goal directed.

3.1 Overview

The reactor is a system, like those discussed by Brooks and others[2, 4, 13], that contains a network of sensory-motor 'linkages' which we will call reactions. The reactor has several key properties:

1. 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. 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 'intelligence' in a reactor is encoded into reactions and cross-connections between them, and annotated with time-constraints.

The reactor is initialized with a number of generally useful behaviors already in place: distinguishing objects from background, locally avoiding obstacles, scanning its sensors around the environment, etc.

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 interacts with the reactor to produce appropriate behavior, in the same way that the reactor interacts with the world (Figure 1). The input to the planner includes: a description of the environment ($\mathcal{E}$) in which the planner is operating; a description of the reactor ($\mathcal{R}$); and advice ($\mathcal{G}$) from a user about objectives the reactor should achieve (e.g., 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 cycles, continually determining from $\mathcal{R}$ if the reactor's behavior in environment $\mathcal{E}$ will indeed conform to the restrictions $\mathcal{G}$. If it determines 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 $\mathcal{G}$. As indicated in Figure 1, there are two connections between planner and reactor:

[1] Our use of this term echos that of Bresina & Drummond.

1. Concurrent Composition: $\mathtt{T} = (\mathtt{P}, \mathtt{Q})^c$. T behaves like P and Q in parallel, connected by port-to-port mapping c. This can also be written as $\mathtt{T} = \Big|_{i \in I} \mathtt{P}^i$. This operator is *idempotent*. Processes are identical if they are concurrent, identically parameterized, and connected instances of the same schema.
2. Sequential Composition: $\mathtt{T} = \mathtt{P}; \mathtt{Q}$. The process T behaves like the process P followed by Q.
3. Conditional Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle : \mathtt{Q}_v$. T behaves like the process P until that terminates. If P aborts, then T aborts. If P stops, then the value v produced by P is acquired, and T then behaves like $\mathtt{Q}_v$.
4. Disabling Composition: $\mathtt{T} = \mathtt{P} \# \mathtt{Q}$. T behaves like the concurrent composition of P and Q until either terminates, then the other is aborted.
5. Synchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :; \mathtt{Q}_v = \mathtt{P} : \mathtt{Q}; \mathtt{P} :; \mathtt{Q}$.
6. Asynchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :: \mathtt{Q}_v = \mathtt{P} : (\mathtt{Q}, \mathtt{P} :: \mathtt{Q})$.

Table 1: $\mathcal{RS}$ Composition Operations.

1. *Perceptions:* Data gathered by the reactor is fed into the planner. By altering the configuration of the reactor, the planner alters what perceptions it receives.
2. *Adaptations:* The planner alters the structure of the reactor by specifying adaptations.

3.2 The Representation of Actions

The $\mathcal{RS}$ model was developed to express robot plans that could 'tune' their behavior to suit their environment. This is particularly relevant in the robot domain because object locations, identities and behaviors are never precisely known in any realistic task. Plans, actions and world models are all represented as hierarchical networks of distributed processes. For a more detailed motivation and description of $\mathcal{RS}$ see [12, 13, 14]; we present the model here from the perspective of its novel representation of actions.

$\mathcal{RS}$ Terminology. An $\mathcal{RS}$ description of a process, or network of processes, is called a *schema*. We shall treat the terms 'process' and 'schema instance' as synonyms. For example, $\mathtt{Joint}_{j=v}$ denotes a process that is an instance of the schema Joint with parameter j set to value v (when unambiguous, we write $\mathtt{Joint}_v$). The 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*. Processes can be defined in terms of networks of other processes, grounding out with a set of atomic, pre-defined, processes. The set of *basic schemas*, B, defines what processes are atomic. The composition operations in Table 1 (for more details see [12, 13]) are used to build plans from elements of B.

Plans, Actions and Worlds. The set B is divided into three classes, $B = S \cup T \cup M$: basic sensory schemas S, basic motor schemas M, and basic task schemas T. Plans are constructed by linking the elements of S to those of M via computations performed by elements of T. That is, every plan is phrased as action linked to sensing. Additionally, since sensing then becomes task-specific, this allows us to represent the world in a fashion similar to Agre & Chapman's *functional-indexical*[1] terms. That is, objects are only represented in $\mathcal{RS}$ in terms of the role they play in plans. Another novel feature of this approach is the following. All actions are represented as processes, and can be sent messages and controlled like any other processes. This is very different from the classical view of actions as atomic events that simply 'happen' and are only characterized by preconditions and effects. Our approach allows us to easily handle effects such as modification of the behavior, or 'inhibition', of actions by other actions, that have proven important in building reactive systems [4, 18].

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: namely networks of concurrent processes. We demonstrate in [13] that it is possible to construct models of uncertain and dynamic worlds in this fashion, and to analyze the concurrent combination of the world and the reactive system to determine their interactive behavior.

Process Evolution. The analysis of programs in process-based models has been under investigation for some time in Computer Science[8, 9]. We follow the techniques of that field in analyzing plans and worlds in $\mathcal{RS}$. It is important to represent how process networks evolve over time, as processes dynamically terminate or are created. In [12, 13] we introduced the evolves operator, which is defined as follows: 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 $\mathtt{A} \xrightarrow{\Omega} \mathtt{B}$. Table 2 summarizes the definition of this operator.

- $\mathtt{P} \xrightarrow{\Omega\mathtt{P}} \mathtt{STOP}$ (definition of termination condition).
- Sequential: $\mathtt{P};\mathtt{Q} \xrightarrow{\Omega\mathtt{P}} \mathtt{Q}$.
- Conditional:
  $\mathtt{P}\langle i\rangle : \mathtt{Q_i} \xrightarrow{\Omega\mathtt{P}} \mathtt{Q_i}$, and also
  $\mathtt{P}\langle i\rangle : \mathtt{Q_i} \longrightarrow \mathtt{ABORT}$ if $\mathtt{P}\langle i\rangle \longrightarrow \mathtt{ABORT}$.
  Different behavior on termination versus abortion is what makes this operator 'conditional.'
- Network: $(\mathtt{P^0},\ldots,\mathtt{P^i},\ldots,\mathtt{P^n}) \longrightarrow (\mathtt{P^i},\ldots,\mathtt{P^n})$ iff $\pi(\Omega\mathtt{P^0},\ldots,\Omega\mathtt{P^{i-1}})$, where $\pi(A,\ldots,B)$ is the disjunction of all orderings (including conjunction) of the conditions $A,\ldots,B$.
- Disabling: $(\mathtt{P^0}\#\ldots\#\mathtt{P^i}\#\ldots\#\mathtt{P^n}) \longrightarrow \mathtt{STOP}$ iff $\Omega\mathtt{P^i}$ for an $i \in \{1\ldots n\}$, in which case for all $j \neq i$ $\mathtt{P^j} \longrightarrow \mathtt{ABORT}$.

STOP is a process that immediately terminates. ABORT is a process that immediately aborts.

Table 2: $\mathcal{RS}$ Evolution Operator.

4 The Planner-Reactor System

This section describes each of the components of the planner-reactor in some detail and finishes with an example of their behavior. The hardware architecture for the implementation is shown in figure 2.

4.1 Reactor

The reactor, and the way in which it can be adapted over time, is specified using the $\mathcal{RS}$ model. The general form of the reactor is the following concurrent network of processes:

$$\mathtt{REACTOR} = (\ \mathtt{E}^1\#\mathtt{P}^1,\ldots,\mathtt{E}^n\#\mathtt{P}^n,\ \mathtt{ADDPLAN}\langle k\rangle :: (\mathtt{E}^k\#\mathtt{P}^k)\ ) \quad (1)$$

where

1. $\mathtt{P}^i$ is a sensory-action pair, a reaction, which will be of the form $\mathtt{Stimulus}\langle q\rangle :; \mathtt{Response}_q$, or part of a plan, which can be of any form, but in our examples will typically be of the form $\mathtt{Locate}_{object}\langle p\rangle : \mathtt{DoAction}_p$.
2. $\mathtt{E}^i$, by the definition of #, is a guard process by which $\mathtt{P}^i$ can be removed by the planner, $\mathtt{E^i} \longrightarrow \mathtt{STOP} \Rightarrow \mathtt{P^i} \longrightarrow \mathtt{ABORT}$.
3. ADDPLAN, by the definition of ::, is the interface through which the planner can add new guarded reactions $\mathtt{E}^m\#\mathtt{P}^m$,

$$\begin{array}{l}\mathtt{ADDPLAN}\langle k\rangle \longrightarrow \mathtt{STOP} \quad \textit{and } k = m \textit{ at termination}, \Rightarrow \\ \mathtt{ADDPLAN}\langle k\rangle :: (\mathtt{E}^k\#\mathtt{P}^k) \longrightarrow \\ \qquad (\mathtt{ADDPLAN}\langle k\rangle :: (\mathtt{E}^k\#\mathtt{P}^k), (\mathtt{E}^m\#\mathtt{P}^m)\ )\end{array} \quad (2)$$

Reactions. The reactor is continuously active, inspecting the world and prepared to act on what it senses. Reactive actions can be easily specified in $\mathcal{RS}$. 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:

$$\mathtt{Locate}_{p1}\langle obj\rangle :; (\mathtt{Alloc}_{\chi,R}\langle r\rangle : \mathtt{Place}_{r,obj,dest})$$

where $\mathtt{Locate}_m$ is a basic sensory process that inspects the world for an instance of part m and terminates when it finds one, producing a pointer to the instance in obj. $\mathtt{Alloc}_{\chi,R}$ allocates a robot[2] according to some constraints χ and only terminates when it gets one, producing a pointer r. Place then uses robot r to acquire and move object instance obj to location $dest$. The recurrent composition ':;' forces Locate to be *continually* recreated. This expresses the conditional goal that *whenever* an instance of $p1$ appears, an attempt is made to get a robot to acquire and move that part.

Plans. Items 2 and 3 above indicate the only ways in which the reactor can be changed by the planner. To facilitate this, plan 'segments' are represented in the reactor in

[2] For more details on on-line allocation of robots, see [11].

Figure 2: System Diagram of the Implementation

the following, *situation-labeled* form:

$$\texttt{P}t = \Big|_{s \in sit(t)} \ (\ \texttt{CheckSit}_s : \texttt{Action}_{t,s} \ \# \ I_{t,s} \ ) \tag{3}$$

as 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 t in which the action (sequence) should be carried out. A situation is a task and world-model specific invariant on the world. $sit(t)$ is the set of situations for task t 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.

4.2 Reactor Adaptation

We want to specify that adaptation of the reactor by the planner must *improve* the behavior of the reactor over time. There are three cases we need to consider:

1. With a given set of goals and given environment, the planner should incrementally adapt the reactor to contain the plans for achieving the given goals in the current environment.
2. Over time, the planner should generalize the applicability of the reactor, i.e., it should adapt the 'plans' in the reactor so that they work under a wider range of conditions.
3. If the goals are changed, the planner should adapt the reactor so that its plans are incrementally changed to suit the new goals; If the environment should change, the planner should incrementally adapt the reactor so that the goals are now achievable in this new environment.

This notion of improvement can be expressed as constraints on the co-evolution of the reactor and world process as they achieve the goal. Let $\texttt{R}^*_{\mathcal{E},\mathcal{G}}$ be the reactor that perfectly achieves the objectives $\mathcal{G}$ in environment $\mathcal{E}$, thus

$$(\ \texttt{R}^*_{\mathcal{E},\mathcal{G}} \ , \ \texttt{W} \ ) \xrightarrow{\omega} \texttt{G} \tag{4}$$

That is, $\texttt{R}^*_{\mathcal{E},\mathcal{G}}$ achieves `G` under constraints ω in world `W`, where `G` and ω constitute the advice $\mathcal{G}$, and `W` is the process model of the environment $\mathcal{E}$. Any specific instance of a reactor $\texttt{R}_{\mathcal{E},\mathcal{G}}$ will be an approximation to the perfect reactor. Item 3 above says that if $\mathcal{E}$ (or $\mathcal{G}$) changes to $\mathcal{E}'$ ($\mathcal{G}'$ respectively) in (4) above, then over time, $\texttt{R}^*_{\mathcal{E},\mathcal{G}}$ should evolve to $\texttt{R}^*_{\mathcal{E}',\mathcal{G}}$ ($\texttt{R}^*_{\mathcal{E},\mathcal{G}'}$ respectively). Items 1 and 2 above say that if $\mathcal{E}$ and $\mathcal{G}$ remain unchanged, then over time $\texttt{R}_{\mathcal{E},\mathcal{G}}$ should evolve towards $\texttt{R}^*_{\mathcal{E},\mathcal{G}}$.

This can be phrased more formally by remembering that the plans in a reactor are represented in the form (3) of situation-labeled action sequences. Ignoring for the moment the `E` and `ADDPLAN` components of the reactor structure, we can say that any reactor $\texttt{R}_{\mathcal{E},\mathcal{G}}$ can be written in terms of the perfect reactor as follows

$$\begin{array}{l} \texttt{R}^* = (\texttt{R} \ , \ \texttt{R}') \\ \textit{where } sit(\texttt{R}^*) = sit(\texttt{R}) \cup sit(\texttt{R}'), \quad \textit{and} \\ s \in sit(\texttt{R}) \cap sit(\texttt{R}') \Rightarrow \texttt{Action}'_s \textit{ extends } \texttt{Action}_s \end{array} \tag{5}$$

One process network extends a second if the second is a prefix of the first. So given two reactors `R1` and `R2` we say `R2` *improves* on `R1` if when they are both written in the form (5) then

1. $sit(\texttt{R2}') \subset sit(\texttt{R1}')$, and
2. for each $s \in sit(\texttt{R1}) \cap sit(\texttt{R2})$ $\texttt{Action2}_s$ *extends* $\texttt{Action1}_s$.

That is, `R2` is a better approximation to $\texttt{R}^*$ that `R1`.

4.3 Planner

Planning is an ongoing activity. Based on the structure of the reactor, the advice and the perceived environment, the planner continually reassesses the appropriateness of the reactors behavior. If this behavior does not suit the advice anymore, either because the advice has changed or the perception of the environment has changed, the planner considers adapting the reactor. Important aspects in our approach to planning for a reactive system are: to plan communication between actions; to allow concurrency of actions; and to continually assess and reassess the appropriateness of the plans. These issues are briefly addressed here.

Planning for communicating actions. The action structure we employ, the $\mathcal{RS}$ process, is a much richer structure than the typical action model used in classical planning. In particular, our actions have the ability to communicate with other actions and accept or send parameters from/to other actions. This allows the planner to plan for *knowledge producing* actions, and have the acquired information be sent to another action that needs it. For example, to start a kitting task, the robot needs to find and place a kit tray. The planner cannot assume that the reactor knows *a priori* where all the instances of trays are located in the work area. However the planner knows it has an action at its disposal, `Locate`, which can determine the location of an object instance. Therefore its plan will consist of the `Place` and `Locate` actions, with an appropriate parameter passing connection between them. In $\mathcal{RS}$ notation this plan is phrased as follows:

$$\texttt{Plan} = \texttt{Locate}_{tray}\langle p \rangle : \texttt{Place}_p$$

The `Locate` action will locate an instance of the type *tray* and once found, it will put the information in the communication port p from which the action `Place` can acquire it.

Another type of communication which has proven important in building reactive systems is communication between concurrently executing actions. For instance, consider the same task as before, but now in a cluttered environment. In that case, it is necessary to use a `Safe_Place` action, a version of `Place` which detours around obstacles. To this end, `Safe_Place` has an input communication port on which it can receive information about the location of obstacles. The process `LocalObstacles` is continuously active in the reactor (say) and provides information about all objects in the vicinity of the robot. The planner produces the following image of what should be in the reactor:

$$\begin{array}{lcl} \texttt{Plan} & = & \texttt{Locate}_{tray}\langle p\rangle \;:\; (\; \texttt{Safe\_Place}_p \;,\; \texttt{LocalObstacles} \;)^c \\ & c: & \texttt{LocalObstacles}(obst) \mapsto \texttt{Safe\_Place}(obst\_info) \end{array} \tag{6}$$

In this plan the `LocalObstacles` action actively looks for local obstacles and as soon as it finds one it puts information about that obstacle on the *obst* port, while continuing to look for others. The `Safe_Place` action can then retrieve this information from its *obst_info* port and use it to carry out local obstacle avoidance.

Planning to adapt the reactor. Planning is a continual process of accepting goals from a user and perceptions from the reactor, and producing adaptations to the reactor. Based on an internal model of the structure of the reactor, the planner continually reassesses the appropriateness of the structure and examines extensions and modifications. New goals or goal modifications will, in general, demand revisions to the reactor. When a new goal is received, the planner simply determines which set of reactions (some of which may be already available in the reactor) are appropriate to conform to the new goal. For this case the planner performs goal-directed reasoning. New perceptions restrict the number of ways the world could evolve and simplify the planner's task. However, if a modification of a goal is required, or a perception is received which conflicts with the world model internal to the planner, the planner has to reason to determine which of the planned reactions are affected. Subsequently, it has to find an appropriate modification for this affected part of the reactor. The plans produced are, in general, not fully refined at once. Often the planner has to settle on an abstract plan of which only the first steps are fully refined. The refinement of the rest of the plan is then suspended until new perceptions are received from knowledge producing actions purposely introduced into the first section of the plan.

The kernel of the planner. Our planner is based on the interval temporal logic (ITL) as described by Pelavin[17]. Formulae in this logic are first order logic predicates augmented with a modal operator @, e.g., $IN(A,D)@i1$ specifies that the predicate $IN(A,D)$ should be evaluated over interval $i1$. In addition the special predicate $ITL($ $interval1$, $itlrelation$, $interval2$) is used to specify constraints between intervals, e.g., $ITL(i1,$ $equal$, $i2)$ specifies that the interval $i1$ should denote the same interval of time as $i2$. Predicates can be of different types, the type specifying how the truth of the predicate relates to the truth of that same predicate over a sub-interval or point (Shoham[20]). In this paper, the only type of predicate we use is a property. For a property it holds that if that property is true over a certain interval, it is true over every subinterval of that interval. e.g., $IN(A,D)@i$ is true whenever there exists an interval $i2$, such that $i2$ encompasses i and $IN(A,D)@i2$ holds. Where *encompasses* is the composite ITL relation (*equals during_i finish_i starts_i*). Actions have executability conditions and effects, both formulas in ITL. 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 (e.g., (6) above), since executability conditions can include conditions that must hold *while* the action is executing.

4.4 Example One

Consider a kitting robot which has the following as part of its current reactor structure:

$$\texttt{P} \;=\; \texttt{Locate}_A\langle r\rangle \;:; (\texttt{Perceive}_r \;,\; \texttt{Place}_{r,buf}) \tag{7}$$

That is, it will look out for any objects in the set A, and each one it finds, it will place into the buffer area buf. The process `Perceive` updates the planners knowledge about the world to include the fact that a new instance of A has arrived. Let `Place` have a port *inh* which can be used to inhibit the process from taking any action. For convenience, let us also assume there is a process `Inhibit` that produces 'inhibit' signals, and can be connected to the *inh* port to inhibit `Place`.

The planner has a model of the purpose of the reactor structure. The ITL representation is as follows:

$$\forall x, i1, i2: \quad Instance(x,A)@i1 \Rightarrow Available(x)@i2 \wedge ITL(i2, equal, i1)$$

This specifies that the reactor maintains availability of an object by placing it in the buffer area.

Now consider the case where the system is given the extra task of responding to the requests of downstream automation for objects in the set $T \subset A$, and with placing instances of those objects on the transport belt. It receives the following additional goal:

$$\forall x, i1, i2: \quad (Instance(x,T) \wedge Request(T))@i1 \Rightarrow Onbelt(x)@i2 \wedge ITL(i1, encompasses, i2)$$

This new goal requires the planner to modify the reactor structure. First, the occurrence of a request has to be determined, and conditional on this the existence of an appropriate object instance, action should be taken to locate and put that instance on the belt. However, since $T \subset A$, this plan of action could interfere with the structure already in place in the reactor, resulting in an attempt to place the same instance both on the belt and in the buffer. Since placing the instance on the belt also maintains availability to the system, an inhibit of the place-in-buffer action resolves the potential interference. After the modifications have been sent to the reactor, the new reactor structure is

$$\begin{array}{lcl} \texttt{P} & = & \texttt{Locate}_A\langle r\rangle \;:;\; (\; \texttt{Perceive}_r \;,\; \texttt{Place}_{r,buf} \;, \\ & & \quad \texttt{Request}\langle T\rangle \;:\; \texttt{IsType}_{r,T} \;: \\ & & \quad\quad (\; \texttt{Inhibit} \;\#\; \texttt{Put}_{r,Belt} \;)\;)^c \\ & c: & \texttt{Inhibit}(inh) \mapsto \texttt{Place}(inh) \end{array} \tag{8}$$

4.5 Example Two

Consider the following kitting robot example: The robot looks for a kit tray t, fills in parts $p1$ and $p2$ (which arrive via a conveyer belt) on that tray, and then sends the tray out of the cell. We will use this example to illustrate how the planner determines what actions to send to the reactor in the case i) that it is constructing a plan from scratch (unusual), and ii) that it is adapting a plan to suit new goals (more usual).

New Goals. The goal is

$$\begin{array}{l} \exists x, y, z, i1, i2[\ (Instance(p1, x) \wedge \\ \quad Instance(p2, y) \wedge Instance(t, z))@i1 \\ \Rightarrow \\ (InJig(z, Jig) \wedge IN(x, z) \wedge \\ \ IN(y, z))@i2 \wedge ITL(i2, finishes, time)\ \ ] \end{array} \tag{9}$$

This conditional goal specifies that when there are instances of $p1$, $p2$ and the tray t available, put the kit together in the jig Jig. The ITL predicate in the consequent assures that, once assembled, the kit stays together by requiring the goal interval $i2$ to finish the special interval $time$, denoting the entire time line. In trying to establish an appropriate reactor structure, the planner first determines the satisfiability of the antecedent of the goal. Since the predicate $Instance$ is of type property, the conjunctive goal is split up into three atomic temporally qualified goals, with intervals completely encompassing the interval $i1$. The antecedent of this is

$$\begin{array}{ll} \exists & x, y, z, i1, i2, i3, i4, i5[(\\ & Instance(p1, x)@i3 \wedge ITL(i3, encompasses, i1) \\ & Instance(p2, y)@i4 \wedge ITL(i4, encompasses, i1) \\ & Instance(t, z)@i5 \wedge ITL(i5, encompasses, i1))] \end{array} \tag{10}$$

Each atomic goal can be achieved by an appropriate `Locate` action. Thus, the consequent of the conditional goal has to be achieved as well. In a similar fashion, this part of the goal is also split up and the appropriate actions are, respectively, `Place,` for placing the tray in the jig and `Insert,` for inserting the components into the tray. Since the tray may not contain components when transported, the place action has to occur before inserting the components. Finally, actions are tagged with the conditions under which they should be executed in the task. Thus the final plan produced is the following (in the situation-labeled form):

$$\begin{array}{llll} \mathtt{Plan} & = & (\ \mathtt{NoTray} & : \ \mathtt{Locate}_t\langle p\rangle : \mathtt{Place}_p\ , \\ & & \ \ \mathtt{Tray}\langle t\rangle & : \ (\ \mathtt{Locate}_{p1}\langle p\rangle : \mathtt{Insert}_{p,t}\ , \\ & & & \quad \mathtt{Locate}_{p2}\langle p\rangle : \mathtt{Insert}_{p,t}\ ) \\ & & \ \ \mathtt{FullTray}\langle t\rangle & : \ \mathtt{Remove}_t\ ) \end{array} \tag{11}$$

This can be added using `ADDPLAN` to the reactor to incrementally produce the following reactor:

$$\begin{array}{lll} \mathtt{REACTOR} & = & (\ \Big|_{i\in 1\ldots 4}\ \mathtt{E}^i\#\mathtt{P}^i\ ,\ \mathtt{ADDPLAN}\langle k\rangle :: (\mathtt{E}^k\#\mathtt{P}^k)\ ) \\ \mathtt{P}^1 & = & \mathtt{NoTray}\ :\ \mathtt{Locate}_t\langle p\rangle : \mathtt{Place}_p\ \#\ \mathtt{I}^1 \\ \mathtt{P}^2 & = & \mathtt{Tray}\langle t\rangle\ :\ \mathtt{Locate}_{p1}\langle p\rangle : \mathtt{Insert}_{p,t}\ \#\ \mathtt{I}^2 \\ \mathtt{P}^3 & = & \mathtt{Tray}\langle t\rangle\ :\ \mathtt{Locate}_{p2}\langle p\rangle : \mathtt{Insert}_{p,t}\ \#\ \mathtt{I}^3 \\ \mathtt{P}^4 & = & \mathtt{FullTray}\langle t\rangle\ :\ \mathtt{Remove}_t\ \#\ \mathtt{I}^4 \end{array} \tag{12}$$

Change of Goal. Now, consider the case when the planner receives a new goal to use $p3$ in place of $p2$. Given this modification on existing advice, e.g. $Modify(Goal, p2 \rightarrow p3)$, where $Goal$ is the previous goal given to the planner, the planner first determines which parts of the plan are affected, by looking at which parts of the plan achieved goals with $p2$. In this case this is `P3` :

$$\mathtt{P}^3 = \mathtt{Tray}\langle t\rangle\ :\ \mathtt{Locate}_{p2}\langle p\rangle : \mathtt{Insert}_{p,t}\ \#\ \mathtt{I}^3 \tag{13}$$

This part of the plan is not appropriate anymore and has to be replaced. The planner traces back the deduction of this part of the plan, and determines what modifications and/or additional actions are necessary to achieve the modified goal. In this case, the new plan part $\mathtt{P}^5$ is very similar to the old one:

$$\mathtt{P}^5 = \mathtt{Tray}\langle t\rangle\ :\ \mathtt{Locate}_{p3}\langle p\rangle : \mathtt{Insert}_{p,t}\ \#\ \mathtt{I}^5 \tag{14}$$

The adaptation of the plan is carried out in two steps. First, the adaptation $\mathtt{P}^5$ is newly defined by the planner. Next, the substitution of this code into the middle of the kitting plan is accomplished by

1. $\mathtt{E}^3$ is terminated by the planner causing $\mathtt{P}^3$ to abort.
2. `ADDPLAN` terminates with $m = 5$, adding the newly-defined $\mathtt{E}^5\#\mathtt{P}^5$ into the reactor.

5 Discussion

In summary, we have presented the kitting robot problem, a practical motivation for building robot systems that can plan and act robustly in an uncertain and dynamic environment. We have discussed why the state of the literature does not yet support building such a system. Our work attacks this problem by casting planning not as an *a priori* activity, but as a continuing adaptation of a reactive system. We use a novel action representation, the $\mathcal{RS}$ model, to facilitate the on-going adaptation of the reactor. The laws by which the planner improves the behavior of the reactor were presented formally. The planner is also unique in that it plans for concurrent communicating actions.

The current status of this work is as follows. The planner and reactor have been tested on the robot workcell for the case where the planner adds in goal-directed actions to an existing reactor. On-going adaptation of the reactor by the planner has not yet been tested. Further theoretical development is necessary, for example, to ensure that the planner-reactor relationship is stable and efficient, and to determine what the appropriate 'units' are by which to incrementally adapt plans. Our initial research towards answering these questions indicates that we have to extend our representation to include probabilistic information. The existing $\mathcal{RS}$ implementation supports real-time scheduling, but, as yet, we have not experimented with time-constrained plans in reacting or planning.

References

[1] P. Agre and D. Chapman. From reaction to participation. In *DARPA Planning workshop*, pages 123–154, Santa Cruz, CA, Oct. 1987.

[2] A. Bestravos, J. Clark, and N. Ferrier. Management of sensori-motor activity in mobile robots. In *IEEE Int. Conf. Robotics & Automation*, pages 592–597, Cincinnati, OH, May 1990.

[3] J. Bresina and M. Drummond. Integrating planning and reaction. In J. Hendler, editor, *AAAI Spring Workshop on Planning in Uncertain, Unpredictable or Changing Environments*, Stanford CA, Mar. 27–29 1990. Systems Research Center, Univ. of Maryland.

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

[5] H. Chochon and R. Alami. Nns, a knowledge-based on-line system for an assembly workcell. In *IEEE Int. Conf. Robotics & Automation*, pages 603–609, San Francisco, CA, 1986.

[6] R.E. Fikes and N.J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[7] B.R. Fox and K.G. Kempf. Reasoning about opportunistic schedules. In *IEEE Int. Conf. Robotics & Automation*, pages 1876–1882, Raleigh, NC, 1987.

[8] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[9] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International Series in Computer Science, 1985.

[10] D. Lyons, S. Mehta, and P. Gopinath. Robust representation and execution of robot plans. In *EuroMicro Real-Time Systems Workshop*, Horsholm, Denmark, June 1990. (Philips TN-89-097).

[11] D.M. Lyons. On-line allocation of robot resources to task plans. In *SPIE Symp. on Advances in Intelligent Robotic Systems; Expert Robots for Industrial Use (Vol. 1008)*, pages 215–222, Nov. 1988.

[12] D.M. Lyons. A Process-Based Approach to Task-Plan Representation. In *IEEE Int. Conf. Rob. & Aut.*, Cincinatti, Ohio, May 1990.

[13] D.M. Lyons. A formal model for reactive robot plans. In *2nd International Conference on Computer Integrated Manufacturing*, RPI, Troy, NY, May 21–23 1990.

[14] 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.

[15] D.M. Lyons and S. Mehta. A distributed computing environment for the multiple robot domain. In *Fourth International Conference on CAD, CAM, Robotics and Factories of the Future*, New Delhi, India, Dec. 19–22 1989.

[16] D.M. Lyons, R. Pelavin, A. Hendriks, and P Benjamin. $\mathcal{RS}$: A formal model for reactive robot plans. In J. Hendler, editor, *AAAI Spring Symposium on Planning in uncertain and changing environments*, Stanford, CA, Mar. 27–29 1990. Systems Research Center, Univ. of Maryland.

[17] R. Pelavin. *A Formal Approach to Planning With Concurrent Actions and External Events.* PhD thesis, Univ. of Rochester, Dept. of Computer SC., Rochester, NY, 1988.

[18] J. Sanborn and J. Hendler. A model of reaction for planning in dynamic environments. *AI in Engineering*, 3(2):95–102, 1988.

[19] C.J. Sellers and S.Y. Nof. Performance analysis of robotic kitting systems. *Rob. & Comp. Integ. Manuf.*, 6(1):15–24, 1989.

[20] Y. Shoham. *Reasoning about change.* MIT Press Series in Artificial Intelligence. MIT Press, 1988.

[21] X. Xiaodong and G.A. Bekey. Sroma: An adaptive scheduler for robotic assembly systems. In *IEEE Int. Conf. Robotics & Automation*, pages 1282–1287, Philadelphia, PA, 1988.

Achieving Robustness by Casting Planning as Adaptation of a Reactive System.

D. M. Lyons, A. J. Hendriks, S. Mehta

Philips Laboratories
North American Philips Corporation
Briarcliff Manor NY 10510

Abstract

Classical AI planning is not sufficiently robust in uncertain and dynamic environments. Reactive approaches are robust in some environments – namely those for which they have been preprogrammed. An approach is presented here that integrates *a priori* planning with reaction. As motivation, a practical robot problem, the kitting robot, is presented. Solving this problem demands a system that can make timely and robust actions in an uncertain environment. A solution to the kitting robot problem is outlined in which planning is cast as adaptation of a reactive system to suit changes in the goals or environment. The reactive system (the *reactor*) is based on a formal model for representing flexible robot plans, the $\mathcal{RS}$ model. Thus, we were able to formalize the mechanisms by which the planner improves the behavior of the reactor. This system was implemented to control a Puma-560 robot equipped with visual sensing.

1 Introduction

Solving the problem of planning and acting robustly in an uncertain environment is a crucial objective in robotics. For example, Fox & Kempf [7] describe how scheduling manufacturing operations suffers from the problem of executional uncertainty: During the execution of a schedule, some of the resources can be expected to become unavailable, but there is no way to tell in advance which one or when. We introduce a practical problem, the kitting robot problem [19], which necessitates producing a robot system that can plan and react in an uncertain environment, and we describe an integrated approach to planning and reacting to address this kind of problem.

Classical AI planning, as exemplified by STRIPS [6] and upon which many planning systems are still based, is not sufficiently robust in uncertain and dynamic environments[17]. Reactive approaches, a more recent development exemplified by Brooks' work[4], operate very robustly in some uncertain and dynamic environments — namely those for which the reactions have been preprogrammed. Our solution to the kitting robot problem improves on the state-of-the-art by combining the ability to plan in advance with the ability to react at *all* times to the environment. We cast planning as *adaptation* of a reactive system (the Reactor) to suit changes in goals or the environment. Rather than generating and then executing plans, our approach is to have the Reactor be a real-time system that continually interacts with the environment, and the planner be a separate and concurrent system that 'tunes' the behavior of the reactor to ensure that goals are achieved. The Reactor is based on a formal framework for representing flexible robot plans, the $\mathcal{RS}$ model[12, 14]. Thus, the behavior of the Reactor, and the rules by which the Reactor can be modified, become open to mathematical analysis. We present a set of rules by which the planner 'improves' the reactor to render its behavior more goal-directed. Adaptation of the reactor by the planner is made possible by the unique way in which we represent actions as processes that can communicate with, and be affected by, other actions. The reactor is implemented on a real-time kernel that directly emulates the $\mathcal{RS}$ model [10, 15]. The planner is implemented under the POPLOG system on a SUN3. The planner-reactor system is used to control a Puma-560 robot equipped with two cameras and a force-sensing, multi-fingered gripper.

Section 2 introduces the kitting problem and describes why the current state of the literature does not support solving it. Section 3 introduces the 'planning as adaptation' concept in more detail, and some background information on our action representation, the $\mathcal{RS}$ model. Section 4 describes each component of the system in more detail, and finishes with an example of the system in operation.

2 Kitting Robot

A kitting robot is a robot system that puts together assembly kits. Simpler and cheaper automation can construct the assemblies once they have been placed in the kits and routed appropriately. Instead of building a factory full of expensive, intelligent robots, the intelligence and cost is focused into a small number of kitting robots which feed the rest of the factory. Kitting provides the line stock availability, the line scheduling flexibility, and the reduced station cycle time productivity for competitive results [19]. However, to achieve this ideal, the kitting robot has to 'iron out' all the uncertainties associated with the assembly process so that it can be done by simpler automation.

The construction of kits is actually a simple form of assembly: there is much less sequentiality (and therefore, more freedom) in placing parts into kits than in assembly, and mating between part and kit is usually not a high-tolerance procedure. The kitting robot has to deal with the following sources of uncertainty:

1. Variable arrival rates and poses of incoming parts. The primary job of the kitting robot is to take parts (perhaps directly from stores) with high position and pose uncertainty, and to put them in a kit so as to reduce the uncertainty.
2. Errors in incoming parts. Faulty parts need to be removed before they either cause errors in downstream automation, or become assembled into (faulty or low-quality) products.
3. Substitutability of incoming parts. That is, which parts can be substituted for other, unavailable parts.
4. Mixed batches of kits. The robot may be serving more than one line, or the lines may be running mixed batches.
5. Variable availability of resources. Examples of resources are tools or machines with which the the robot needs to coordinate directly.
6. Response to events in the downstream automation. For example, machines breaking down, or alternate machine configurations.
7. Response to factory management advice. For example, response to changing the batch mix or throughput, or implementing once-off fixes for special situations.

Thus, 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 dynamic environment.

In order to deal with events as they happen, the kitting robot needs to be capable of *real-time, reactive response*. On the other hand, in order 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'*. The current literature in Robotics and AI does not yet support building a robot control system that successfully incorporates both of these attributes.

Classical AI planners, e.g., STRIPS[6], take one 'snapshot' of the world, and produce a fixed sequence of actions to achieve a goal in that world. This approach is too inflexible to work in an uncertain and dynamic environment: by the time the plan has been produced, the world may well have changed in such a fashion as to render the plan useless. There is an even more important limitation with this approach: STRIPS-like, state-based approaches make it almost impossible to represent *concurrent actions*, and the actions of *external entities* (entities other than the active agent, such as other machines, or even the forces of nature)[17]. Despite this, such planners are still common [?].

More recently, the concept of systems that react, rather than plan, has been introduced by Brooks [4] and Agre & Chapman [1] among others. This approach allows for fast and flexible response to an environment that can be complex, dynamic and uncertain. One example of the advantage of such a reactive system in an industrial setting is contained in Fox & Kempf's paper[7] on opportunistic scheduling.

There are two basic problems with the purely reactive approach: Firstly, to date the building of reactive systems has been largely manual and *ad-hoc*. We argue here that this is inappropriate. A reactive system can be expected to exhibit complex behavior in a complex environment[4]; that is one of the key advantages of the approach. However, human intuition is notoriously unreliable in predicting such complex situations. We should make every effort to use mathematical techniques for predicting and designing reactive behavior if at all possible. This was our motivation in previous work [12, 16]. Bestavros et al.[2] also describe a formal framework for building reactive systems, with many similarities to our approach.

The second problem is that a reactive approach doesn't work well for situations outside the range for which the 'reactions' were programmed[18]. There have been a number of attempts to combine the ability to react with that of *a priori* planning (usually in the context of an assembly workcell) [5, 21]. In general, these systems work by generating a complete plan in advance. This plan is then down-loaded to an Executor or Scheduler, which then chooses an appropriate action sequence based on the state of the environment. This misses an important point however. If a reactive system is indeed called for in the problem specification, then the standard planner-executor division will simply not suffice for two reasons:

1. Generation and loading of entire plans is unrealistic in a dynamic and uncertain environment.
2. An executor (or scheduler) can only act when it has been loaded with a plan; a reactive system needs to be able to act at any time.

In the remainder of this paper, we will describe a novel approach to integrating planning and reaction in the kitting robot domain. The approach has the following characteristics.

1. It is based on a precise framework for reactive plans, the $\mathcal{RS}$ model, and formal techniques for looking at the behavior of plans situated in their environment.
2. It defines planning not as the off-line *generation* of plans, but as the on-line *adaptation* of a reactive system to suit the environment and system goals.

3 Planning and Reacting

The usual definition of a planner is a program that produces a sequence of operations that when applied to the world, will take the world from its initial state to a goal state. Typically, this plan is then sent to an executer which carries out each of the actions. Our definition of planning is more akin to those of Agre & Chapman [1] or Bresina &

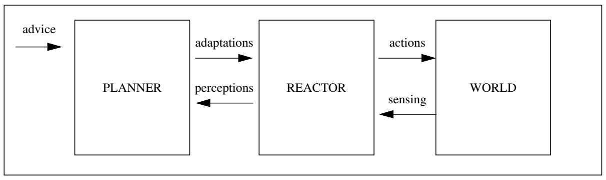

Figure 1: Planner-Reactor-World System

Drummond[3]: a planner is a system that continually modifies a reactive system, the reactor[1], so that its behavior becomes more goal directed.

3.1 Overview

The reactor is a system, like those discussed by Brooks and others[2, 4, 13], that contains a network of sensory-motor 'linkages' which we will call reactions. The reactor has several key properties:

1. 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. 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 'intelligence' in a reactor is encoded into reactions and cross-connections between them, and annotated with time-constraints.

The reactor is initialized with a number of generally useful behaviors already in place: distinguishing objects from background, locally avoiding obstacles, scanning its sensors around the environment, etc.

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 interacts with the reactor to produce appropriate behavior, in the same way that the reactor interacts with the world (Figure 1). The input to the planner includes: a description of the environment ($\mathcal{E}$) in which the planner is operating; a description of the reactor ($\mathcal{R}$); and advice ($\mathcal{G}$) from a user about objectives the reactor should achieve (e.g., 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 cycles, continually determining from $\mathcal{R}$ if the reactor's behavior in environment $\mathcal{E}$ will indeed conform to the restrictions $\mathcal{G}$. If it determines 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 $\mathcal{G}$. As indicated in Figure 1, there are two connections between planner and reactor:

[1] Our use of this term echos that of Bresina & Drummond.

1. Concurrent Composition: $\mathtt{T} = (\mathtt{P}, \mathtt{Q})^c$. T behaves like P and Q in parallel, connected by port-to-port mapping c. This can also be written as $\mathtt{T} = \Big|_{i \in I} \mathtt{P}^i$. This operator is *idempotent*. Processes are identical if they are concurrent, identically parameterized, and connected instances of the same schema.
2. Sequential Composition: $\mathtt{T} = \mathtt{P}; \mathtt{Q}$. The process T behaves like the process P followed by Q.
3. Conditional Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle : \mathtt{Q}_v$. T behaves like the process P until that terminates. If P aborts, then T aborts. If P stops, then the value v produced by P is acquired, and T then behaves like $\mathtt{Q}_v$.
4. Disabling Composition: $\mathtt{T} = \mathtt{P}\#\mathtt{Q}$. T behaves like the concurrent composition of P and Q until either terminates, then the other is aborted.
5. Synchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :; \mathtt{Q}_v = \mathtt{P} : \mathtt{Q}; \mathtt{P} :; \mathtt{Q}$.
6. Asynchronous Recurrent Composition: $\mathtt{T} = \mathtt{P}\langle v \rangle :: \mathtt{Q}_v = \mathtt{P} : (\mathtt{Q}, \mathtt{P} :: \mathtt{Q})$.

Table 1: $\mathcal{RS}$ Composition Operations.

1. *Perceptions:* Data gathered by the reactor is fed into the planner. By altering the configuration of the reactor, the planner alters what perceptions it receives.
2. *Adaptations:* The planner alters the structure of the reactor by specifying adaptations.

3.2 The Representation of Actions

The $\mathcal{RS}$ model was developed to express robot plans that could 'tune' their behavior to suit their environment. This is particularly relevant in the robot domain because object locations, identities and behaviors are never precisely known in any realistic task. Plans, actions and world models are all represented as hierarchical networks of distributed processes. For a more detailed motivation and description of $\mathcal{RS}$ see [12, 13, 14]; we present the model here from the perspective of its novel representation of actions.

$\mathcal{RS}$ Terminology. An $\mathcal{RS}$ description of a process, or network of processes, is called a *schema*. We shall treat the terms 'process' and 'schema instance' as synonyms. For example, $\mathtt{Joint}_{j=v}$ denotes a process that is an instance of the schema Joint with parameter j set to value v (when unambiguous, we write $\mathtt{Joint}_v$). The 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*. Processes can be defined in terms of networks of other processes, grounding out with a set of atomic, pre-defined, processes. The set of *basic schemas*, B, defines what processes are atomic. The composition operations in Table 1 (for more details see [12, 13]) are used to build plans from elements of B.

Plans, Actions and Worlds. The set B is divided into three classes, $B = S \cup T \cup M$: basic sensory schemas S, basic motor schemas M, and basic task schemas T. Plans are constructed by linking the elements of S to those of M via computations performed by elements of T. That is, every plan is phrased as action linked to sensing. Additionally, since sensing then becomes task-specific, this allows us to represent the world in a fashion similar to Agre & Chapman's *functional-indexical*[1] terms. That is, objects are only represented in $\mathcal{RS}$ in terms of the role they play in plans. Another novel feature of this approach is the following. All actions are represented as processes, and can be sent messages and controlled like any other processes. This is very different from the classical view of actions as atomic events that simply 'happen' and are only characterized by preconditions and effects. Our approach allows us to easily handle effects such as modification of the behavior, or 'inhibition', of actions by other actions, that have proven important in building reactive systems [4, 18].

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: namely networks of concurrent processes. We demonstrate in [13] that it is possible to construct models of uncertain and dynamic worlds in this fashion, and to analyze the concurrent combination of the world and the reactive system to determine their interactive behavior.

Process Evolution. The analysis of programs in process-based models has been under investigation for some time in Computer Science[8, 9]. We follow the techniques of that field in analyzing plans and worlds in $\mathcal{RS}$. It is important to represent how process networks evolve over time, as processes dynamically terminate or are created. In [12, 13] we introduced the evolves operator, which is defined as follows: 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 $\mathtt{A} \xrightarrow{\Omega} \mathtt{B}$. Table 2 summarizes the definition of this operator.

- $\mathtt{P} \xrightarrow{\Omega\mathtt{P}} \mathtt{STOP}$ (definition of termination condition).
- Sequential: $\mathtt{P};\mathtt{Q} \xrightarrow{\Omega\mathtt{P}} \mathtt{Q}$.
- Conditional:
  $\mathtt{P}\langle i\rangle : \mathtt{Q_i} \xrightarrow{\Omega\mathtt{P}} \mathtt{Q_i}$, and also
  $\mathtt{P}\langle i\rangle : \mathtt{Q_i} \longrightarrow \mathtt{ABORT}$ if $\mathtt{P}\langle i\rangle \longrightarrow \mathtt{ABORT}$.
  Different behavior on termination versus abortion is what makes this operator 'conditional.'
- Network: $(\mathtt{P^0},\ldots,\mathtt{P^i},\ldots,\mathtt{P^n}) \longrightarrow (\mathtt{P^i},\ldots,\mathtt{P^n})$ iff $\pi(\Omega\mathtt{P^0},\ldots,\Omega\mathtt{P^{i-1}})$, where $\pi(A,\ldots,B)$ is the disjunction of all orderings (including conjunction) of the conditions $A,\ldots,B$.
- Disabling: $(\mathtt{P^0}\#\ldots\#\mathtt{P^i}\#\ldots\#\mathtt{P^n}) \longrightarrow \mathtt{STOP}$ iff $\Omega\mathtt{P^i}$ for an $i \in \{1\ldots n\}$, in which case for all $j \neq i$ $\mathtt{P^j} \longrightarrow \mathtt{ABORT}$.

STOP is a process that immediately terminates. ABORT is a process that immediately aborts.

Table 2: $\mathcal{RS}$ Evolution Operator.

4 The Planner-Reactor System

This section describes each of the components of the planner-reactor in some detail and finishes with an example of their behavior. The hardware architecture for the implementation is shown in figure 2.

4.1 Reactor

The reactor, and the way in which it can be adapted over time, is specified using the $\mathcal{RS}$ model. The general form of the reactor is the following concurrent network of processes:

$$\mathtt{REACTOR} = (\ \mathtt{E}^1\#\mathtt{P}^1,\ldots,\mathtt{E}^n\#\mathtt{P}^n,\ \mathtt{ADDPLAN}\langle k\rangle :: (\mathtt{E}^k\#\mathtt{P}^k)\ ) \quad (1)$$

where

1. $\mathtt{P}^i$ is a sensory-action pair, a reaction, which will be of the form $\mathtt{Stimulus}\langle q\rangle :; \mathtt{Response}_q$, or part of a plan, which can be of any form, but in our examples will typically be of the form $\mathtt{Locate}_{object}\langle p\rangle : \mathtt{DoAction}_p$.
2. $\mathtt{E}^i$, by the definition of #, is a guard process by which $\mathtt{P}^i$ can be removed by the planner, $\mathtt{E^i} \longrightarrow \mathtt{STOP} \Rightarrow \mathtt{P^i} \longrightarrow \mathtt{ABORT}$.
3. ADDPLAN, by the definition of ::, is the interface through which the planner can add new guarded reactions $\mathtt{E}^m\#\mathtt{P}^m$,

$$\begin{array}{l}\mathtt{ADDPLAN}\langle k\rangle \longrightarrow \mathtt{STOP} \quad \textit{and } k = m \textit{ at termination}, \Rightarrow \\ \mathtt{ADDPLAN}\langle k\rangle :: (\mathtt{E}^k\#\mathtt{P}^k) \longrightarrow \\ \qquad (\mathtt{ADDPLAN}\langle k\rangle :: (\mathtt{E}^k\#\mathtt{P}^k), (\mathtt{E}^m\#\mathtt{P}^m)\ )\end{array} \quad (2)$$

Reactions. The reactor is continuously active, inspecting the world and prepared to act on what it senses. Reactive actions can be easily specified in $\mathcal{RS}$. 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:

$$\mathtt{Locate}_{p1}\langle obj\rangle :; (\mathtt{Alloc}_{\chi,R}\langle r\rangle : \mathtt{Place}_{r,obj,dest})$$

where $\mathtt{Locate}_m$ is a basic sensory process that inspects the world for an instance of part m and terminates when it finds one, producing a pointer to the instance in obj. $\mathtt{Alloc}_{\chi,R}$ allocates a robot[2] according to some constraints χ and only terminates when it gets one, producing a pointer r. Place then uses robot r to acquire and move object instance obj to location $dest$. The recurrent composition ':;' forces Locate to be *continually* recreated. This expresses the conditional goal that *whenever* an instance of $p1$ appears, an attempt is made to get a robot to acquire and move that part.

Plans. Items 2 and 3 above indicate the only ways in which the reactor can be changed by the planner. To facilitate this, plan 'segments' are represented in the reactor in

[2]For more details on on-line allocation of robots, see [11].

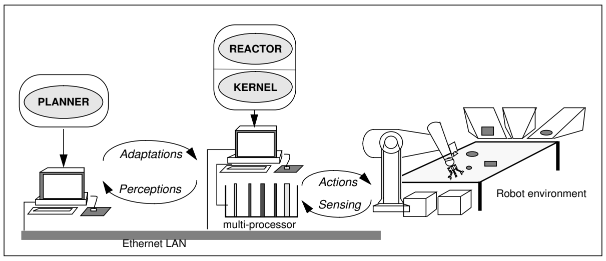

Figure 2: System Diagram of the Implementation

the following, *situation-labeled* form:

$$\mathtt{Pt} = \mathop{\Big|}_{s \in sit(t)} (\ \mathtt{CheckSit}_s : \mathtt{Action}_{t,s} \ \# \ I_{t,s}\ ) \qquad (3)$$

as 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 t in which the action (sequence) should be carried out. A situation is a task and world-model specific invariant on the world. $sit(t)$ is the set of situations for task t 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.

4.2 Reactor Adaptation

We want to specify that adaptation of the reactor by the planner must *improve* the behavior of the reactor over time. There are three cases we need to consider:

1. With a given set of goals and given environment, the planner should incrementally adapt the reactor to contain the plans for achieving the given goals in the current environment.
2. Over time, the planner should generalize the applicability of the reactor, i.e., it should adapt the 'plans' in the reactor so that they work under a wider range of conditions.
3. If the goals are changed, the planner should adapt the reactor so that its plans are incrementally changed to suit the new goals; If the environment should change, the planner should incrementally adapt the reactor so that the goals are now achievable in this new environment.

This notion of improvement can be expressed as constraints on the co-evolution of the reactor and world process as they achieve the goal. Let $\mathtt{R}^*_{\mathcal{E},\mathcal{G}}$ be the reactor that perfectly achieves the objectives $\mathcal{G}$ in environment $\mathcal{E}$, thus

$$(\ \mathtt{R}^*_{\mathcal{E},\mathcal{G}}\ ,\ \mathtt{W}\ ) \xrightarrow{\omega} \mathtt{G} \qquad (4)$$

That is, $\mathtt{R}^*_{\mathcal{E},\mathcal{G}}$ achieves `G` under constraints ω in world `W`, where `G` and ω constitute the advice $\mathcal{G}$, and `W` is the process model of the environment $\mathcal{E}$. Any specific instance of a reactor $\mathtt{R}_{\mathcal{E},\mathcal{G}}$ will be an approximation to the perfect reactor. Item 3 above says that if $\mathcal{E}$ (or $\mathcal{G}$) changes to $\mathcal{E}'$ ($\mathcal{G}'$ respectively) in (4) above, then over time, $\mathtt{R}^*_{\mathcal{E},\mathcal{G}}$ should evolve to $\mathtt{R}^*_{\mathcal{E}',\mathcal{G}}$ ($\mathtt{R}^*_{\mathcal{E},\mathcal{G}'}$ respectively). Items 1 and 2 above say that if $\mathcal{E}$ and $\mathcal{G}$ remain unchanged, then over time $\mathtt{R}_{\mathcal{E},\mathcal{G}}$ should evolve towards $\mathtt{R}^*_{\mathcal{E},\mathcal{G}}$.

This can be phrased more formally by remembering that the plans in a reactor are represented in the form (3) of situation-labeled action sequences. Ignoring for the moment the `E` and `ADDPLAN` components of the reactor structure, we can say that any reactor $\mathtt{R}_{\mathcal{E},\mathcal{G}}$ can be written in terms of the perfect reactor as follows

$$\begin{array}{l} \mathtt{R}^* = (\mathtt{R}\ ,\ \mathtt{R}') \\ \textit{where } sit(\mathtt{R}^*) = sit(\mathtt{R}) \cup sit(\mathtt{R}'), \quad \textit{and} \\ s \in sit(\mathtt{R}) \cap sit(\mathtt{R}') \Rightarrow \mathtt{Action'}_s \ \textit{extends}\ \mathtt{Action}_s \end{array} \qquad (5)$$

One process network extends a second if the second is a prefix of the first. So given two reactors `R1` and `R2` we say `R2` *improves* on `R1` if when they are both written in the form (5) then

1. $sit(\mathtt{R2}') \subset sit(\mathtt{R1}')$, and
2. for each $s \in sit(\mathtt{R1}) \cap sit(\mathtt{R2})$ $\mathtt{Action2}_s$ *extends* $\mathtt{Action1}_s$.

That is, `R2` is a better approximation to $\mathtt{R}^*$ that `R1`.

4.3 Planner

Planning is an ongoing activity. Based on the structure of the reactor, the advice and the perceived environment, the planner continually reassesses the appropriateness of the reactors behavior. If this behavior does not suit the advice anymore, either because the advice has changed or the perception of the environment has changed, the planner considers adapting the reactor. Important aspects in our approach to planning for a reactive system are: to plan communication between actions; to allow concurrency of actions; and to continually assess and reassess the appropriateness of the plans. These issues are briefly addressed here.

Planning for communicating actions. The action structure we employ, the $\mathcal{RS}$ process, is a much richer structure than the typical action model used in classical planning. In particular, our actions have the ability to communicate with other actions and accept or send parameters from/to other actions. This allows the planner to plan for *knowledge producing* actions, and have the acquired information be sent to another action that needs it. For example, to start a kitting task, the robot needs to find and place a kit tray. The planner cannot assume that the reactor knows *a priori* where all the instances of trays are located in the work area. However the planner knows it has an action at its disposal, `Locate`, which can determine the location of an object instance. Therefore its plan will consist of the `Place` and `Locate` actions, with an appropriate parameter passing connection between them. In $\mathcal{RS}$ notation this plan is phrased as follows:

$$\mathtt{Plan} = \mathtt{Locate}_{tray}\langle p\rangle : \mathtt{Place}_p$$

The `Locate` action will locate an instance of the type *tray* and once found, it will put the information in the communication port p from which the action `Place` can acquire it.

Another type of communication which has proven important in building reactive systems is communication between concurrently executing actions. For instance, consider the same task as before, but now in a cluttered environment. In that case, it is necessary to use a `Safe_Place` action, a version of `Place` which detours around obstacles. To this end, `Safe_Place` has an input communication port on which it can receive information about the location of obstacles. The process `LocalObstacles` is continuously active in the reactor (say) and provides information about all objects in the vicinity of the robot. The planner produces the following image of what should be in the reactor:

$$\begin{aligned} \texttt{Plan} \quad = \quad & \texttt{Locate}_{tray}\langle p\rangle \ : \\ & \quad (\ \texttt{Safe\_Place}_p \ , \ \texttt{LocalObstacles}\ )^c \\ c: \quad & \texttt{LocalObstacles}(obst) \mapsto \texttt{Safe\_Place}(obst\_info) \end{aligned} \tag{6}$$

In this plan the `LocalObstacles` action actively looks for local obstacles and as soon as it finds one it puts information about that obstacle on the *obst* port, while continuing to look for others. The `Safe_Place` action can then retrieve this information from its $obst\_info$ port and use it to carry out local obstacle avoidance.

Planning to adapt the reactor. Planning is a continual process of accepting goals from a user and perceptions from the reactor, and producing adaptations to the reactor. Based on an internal model of the structure of the reactor, the planner continually reassesses the appropriateness of the structure and examines extensions and modifications. New goals or goal modifications will, in general, demand revisions to the reactor. When a new goal is received, the planner simply determines which set of reactions (some of which may be already available in the reactor) are appropriate to conform to the new goal. For this case the planner performs goal-directed reasoning. New perceptions restrict the number of ways the world could evolve and simplify the planner's task. However, if a modification of a goal is required, or a perception is received which conflicts with the world model internal to the planner, the planner has to reason to determine which of the planned reactions are affected. Subsequently, it has to find an appropriate modification for this affected part of the reactor. The plans produced are, in general, not fully refined at once. Often the planner has to settle on an abstract plan of which only the first steps are fully refined. The refinement of the rest of the plan is then suspended until new perceptions are received from knowledge producing actions purposely introduced into the first section of the plan.

The kernel of the planner. Our planner is based on the interval temporal logic (ITL) as described by Pelavin[17]. Formulae in this logic are first order logic predicates augmented with a modal operator @, e.g., $IN(A,D)@i1$ specifies that the predicate $IN(A,D)$ should be evaluated over interval $i1$. In addition the special predicate $ITL($ $interval1$, $itlrelation$, $interval2$) is used to specify constraints between intervals, e.g., $ITL(i1,\ equal,\ i2)$ specifies that the interval $i1$ should denote the same interval of time as $i2$. Predicates can be of different types, the type specifying how the truth of the predicate relates to the truth of that same predicate over a sub-interval or point (Shoham[20]). In this paper, the only type of predicate we use is a property. For a property it holds that if that property is true over a certain interval, it is true over every subinterval of that interval. e.g., $IN(A,D)@i$ is true whenever there exists an interval $i2$, such that $i2$ encompasses i and $IN(A,D)@i2$ holds. Where *encompasses* is the composite ITL relation (*equals during_i finish_i starts_i*). Actions have executability conditions and effects, both formulas in ITL. 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 (e.g., (6) above), since executability conditions can include conditions that must hold *while* the action is executing.

4.4 Example One

Consider a kitting robot which has the following as part of its current reactor structure:

$$\texttt{P} \quad = \quad \texttt{Locate}_A\langle r\rangle ::(\texttt{Perceive}_r \ , \ \texttt{Place}_{r,buf}) \tag{7}$$

That is, it will look out for any objects in the set A, and each one it finds, it will place into the buffer area buf. The process `Perceive` updates the planners knowledge about the world to include the fact that a new instance of A has arrived. Let `Place` have a port inh which can be used to inhibit the process from taking any action. For convenience, let us also assume there is a process `Inhibit` that produces 'inhibit' signals, and can be connected to the inh port to inhibit `Place`.

The planner has a model of the purpose of the reactor structure. The ITL representation is as follows:

$$\begin{aligned} \forall x, i1, i2: \quad & Instance(x,A)@i1 \Rightarrow \\ & Available(x)@i2 \wedge ITL(i2, equal, i1) \end{aligned}$$

This specifies that the reactor maintains availability of an object by placing it in the buffer area.

Now consider the case where the system is given the extra task of responding to the requests of downstream automation for objects in the set $T \subset A$, and with placing instances of those objects on the transport belt. It receives the following additional goal:

$$\begin{aligned} \forall x, i1, i2: \quad & (Instance(x,T) \wedge Request(T))@i1 \\ & \Rightarrow \\ & Onbelt(x)@i2 \wedge ITL(i1, encompasses, i2) \end{aligned}$$

This new goal requires the planner to modify the reactor structure. First, the occurrence of a request has to be determined, and conditional on this the existence of an appropriate object instance, action should be taken to locate and put that instance on the belt. However, since $T \subset A$, this plan of action could interfere with the structure already in place in the reactor, resulting in an attempt to place the same instance both on the belt and in the buffer. Since placing the instance on the belt also maintains availability to the system, an inhibit of the place-in-buffer action resolves the potential interference. After the modifications have been sent to the reactor, the new reactor structure is

$$\begin{aligned} \texttt{P} \quad = \quad & \texttt{Locate}_A\langle r\rangle \ :: \ (\ \texttt{Perceive}_r \ , \ \texttt{Place}_{r,buf} \ , \\ & \quad \texttt{Request}\langle T\rangle \ : \ \texttt{IsType}_{r,T} \ : \\ & \quad\quad (\ \texttt{Inhibit} \ \# \ \texttt{Put}_{r,Belt}\ )\ )^c \\ c: \quad & \texttt{Inhibit}(inh) \mapsto \texttt{Place}(inh) \end{aligned} \tag{8}$$

4.5 Example Two

Consider the following kitting robot example: The robot looks for a kit tray t, fills in parts $p1$ and $p2$ (which arrive via a conveyer belt) on that tray, and then sends the tray out of the cell. We will use this example to illustrate how the planner determines what actions to send to the reactor in the case i) that it is constructing a plan from scratch (unusual), and ii) that it is adapting a plan to suit new goals (more usual).

New Goals. The goal is

$$\begin{aligned}&\exists x, y, z, i1, i2[\ (Instance(p1, x)\wedge \\ &\quad Instance(p2, y) \wedge Instance(t, z))@i1 \\ &\Rightarrow \\ &(InJig(z, Jig) \wedge IN(x, z)\wedge \\ &\ IN(y, z))@i2 \wedge ITL(i2, finishes, time)\ \ ]\end{aligned} \tag{9}$$

This conditional goal specifies that when there are instances of $p1$, $p2$ and the tray t available, put the kit together in the jig Jig. The ITL predicate in the consequent assures that, once assembled, the kit stays together by requiring the goal interval $i2$ to finish the special interval $time$, denoting the entire time line. In trying to establish an appropriate reactor structure, the planner first determines the satisfiability of the antecedent of the goal. Since the predicate $Instance$ is of type property, the conjunctive goal is split up into three atomic temporally qualified goals, with intervals completely encompassing the interval $i1$. The antecedent of this is

$$\begin{aligned}\exists\quad & x, y, z, i1, i2, i3, i4, i5[(\\ & Instance(p1, x)@i3 \wedge ITL(i3, encompasses, i1) \\ & Instance(p2, y)@i4 \wedge ITL(i4, encompasses, i1) \\ & Instance(t, z)@i5 \wedge ITL(i5, encompasses, i1))]\end{aligned} \tag{10}$$

Each atomic goal can be achieved by an appropriate `Locate` action. Thus, the consequent of the conditional goal has to be achieved as well. In a similar fashion, this part of the goal is also split up and the appropriate actions are, respectively, `Place,` for placing the tray in the jig and `Insert,` for inserting the components into the tray. Since the tray may not contain components when transported, the place action has to occur before inserting the components. Finally, actions are tagged with the conditions under which they should be executed in the task. Thus the final plan produced is the following (in the situation-labeled form):

$$\begin{array}{llll}\texttt{Plan} & = & (\ \texttt{NoTray} & : \ \texttt{Locate}_t\langle p\rangle : \texttt{Place}_p\ , \\ & & \ \ \texttt{Tray}\langle t\rangle & : \ (\ \texttt{Locate}_{p1}\langle p\rangle : \texttt{Insert}_{p,t}\ , \\ & & & \quad \texttt{Locate}_{p2}\langle p\rangle : \texttt{Insert}_{p,t}\ ) \\ & & \ \ \texttt{FullTray}\langle t\rangle & : \ \texttt{Remove}_t\ )\end{array} \tag{11}$$

This can be added using `ADDPLAN` to the reactor to incrementally produce the following reactor:

$$\begin{array}{lll}\texttt{REACTOR} & = & (\ \Big|_{i\in 1\ldots 4}\ \texttt{E}^i\#\texttt{P}^i\ ,\ \texttt{ADDPLAN}\langle k\rangle :: (\texttt{E}^k\#\texttt{P}^k)\ ) \\ \texttt{P}^1 & = & \texttt{NoTray}\ :\ \texttt{Locate}_t\langle p\rangle : \texttt{Place}_p\ \#\ \texttt{I}^1 \\ \texttt{P}^2 & = & \texttt{Tray}\langle t\rangle\ :\ \texttt{Locate}_{p1}\langle p\rangle : \texttt{Insert}_{p,t}\ \#\ \texttt{I}^2 \\ \texttt{P}^3 & = & \texttt{Tray}\langle t\rangle\ :\ \texttt{Locate}_{p2}\langle p\rangle : \texttt{Insert}_{p,t}\ \#\ \texttt{I}^3 \\ \texttt{P}^4 & = & \texttt{FullTray}\langle t\rangle\ :\ \texttt{Remove}_t\ \#\ \texttt{I}^4\end{array} \tag{12}$$

Change of Goal. Now, consider the case when the planner receives a new goal to use $p3$ in place of $p2$. Given this modification on existing advice, e.g. $Modify(Goal, p2 \rightarrow p3)$, where $Goal$ is the previous goal given to the planner, the planner first determines which parts of the plan are affected, by looking at which parts of the plan achieved goals with $p2$. In this case this is `P3` :

$$\texttt{P}^3 = \texttt{Tray}\langle t\rangle\ :\ \texttt{Locate}_{p2}\langle p\rangle : \texttt{Insert}_{p,t}\ \#\ \texttt{I}^3 \tag{13}$$

This part of the plan is not appropriate anymore and has to be replaced. The planner traces back the deduction of this part of the plan, and determines what modifications and/or additional actions are necessary to achieve the modified goal. In this case, the new plan part $\texttt{P}^5$ is very similar to the old one:

$$\texttt{P}^5 = \texttt{Tray}\langle t\rangle\ :\ \texttt{Locate}_{p3}\langle p\rangle : \texttt{Insert}_{p,t}\ \#\ \texttt{I}^5 \tag{14}$$

The adaptation of the plan is carried out in two steps. First, the adaptation $\texttt{P}^5$ is newly defined by the planner. Next, the substitution of this code into the middle of the kitting plan is accomplished by

1. $\texttt{E}^3$ is terminated by the planner causing $\texttt{P}^3$ to abort.
2. `ADDPLAN` terminates with $m = 5$, adding the newly-defined $\texttt{E}^5\#\texttt{P}^5$ into the reactor.

5 Discussion

In summary, we have presented the kitting robot problem, a practical motivation for building robot systems that can plan and act robustly in an uncertain and dynamic environment. We have discussed why the state of the literature does not yet support building such a system. Our work attacks this problem by casting planning not as an *a priori* activity, but as a continuing adaptation of a reactive system. We use a novel action representation, the $\mathcal{RS}$ model, to facilitate the on-going adaptation of the reactor. The laws by which the planner improves the behavior of the reactor were presented formally. The planner is also unique in that it plans for concurrent communicating actions.

The current status of this work is as follows. The planner and reactor have been tested on the robot workcell for the case where the planner adds in goal-directed actions to an existing reactor. On-going adaptation of the reactor by the planner has not yet been tested. Further theoretical development is necessary, for example, to ensure that the planner-reactor relationship is stable and efficient, and to determine what the appropriate 'units' are by which to incrementally adapt plans. Our initial research towards answering these questions indicates that we have to extend our representation to include probabilistic information. The existing $\mathcal{RS}$ implementation supports real-time scheduling, but, as yet, we have not experimented with time-constrained plans in reacting or planning.

References

[1] P. Agre and D. Chapman. From reaction to participation. In *DARPA Planning workshop*, pages 123–154, Santa Cruz, CA, Oct. 1987.

[2] A. Bestravos, J. Clark, and N. Ferrier. Management of sensori-motor activity in mobile robots. In *IEEE Int. Conf. Robotics & Automation*, pages 592–597, Cincinnati, OH, May 1990.

[3] J. Bresina and M. Drummond. Integrating planning and reaction. In J. Hendler, editor, *AAAI Spring Workshop on Planning in Uncertain, Unpredictable or Changing Environments*, Stanford CA, Mar. 27–29 1990. Systems Research Center, Univ. of Maryland.

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

[5] H. Chochon and R. Alami. Nns, a knowledge-based on-line system for an assembly workcell. In *IEEE Int. Conf. Robotics & Automation*, pages 603–609, San Francisco, CA, 1986.

[6] R.E. Fikes and N.J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[7] B.R. Fox and K.G. Kempf. Reasoning about opportunistic schedules. In *IEEE Int. Conf. Robotics & Automation*, pages 1876–1882, Raleigh, NC, 1987.

[8] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[9] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International Series in Computer Science, 1985.

[10] D. Lyons, S. Mehta, and P. Gopinath. Robust representation and execution of robot plans. In *EuroMicro Real-Time Systems Workshop*, Horsholm, Denmark, June 1990. (Philips TN-89-097).

[11] D.M. Lyons. On-line allocation of robot resources to task plans. In *SPIE Symp. on Advances in Intelligent Robotic Systems; Expert Robots for Industrial Use (Vol. 1008)*, pages 215–222, Nov. 1988.

[12] D.M. Lyons. A Process-Based Approach to Task-Plan Representation. In *IEEE Int. Conf. Rob. & Aut.*, Cincinatti, Ohio, May 1990.

[13] D.M. Lyons. A formal model for reactive robot plans. In *2nd International Conference on Computer Integrated Manufacturing*, RPI, Troy, NY, May 21–23 1990.

[14] 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.

[15] D.M. Lyons and S. Mehta. A distributed computing environment for the multiple robot domain. In *Fourth International Conference on CAD, CAM, Robotics and Factories of the Future*, New Delhi, India, Dec. 19–22 1989.

[16] D.M. Lyons, R. Pelavin, A. Hendriks, and P Benjamin. $\mathcal{RS}$: A formal model for reactive robot plans. In J. Hendler, editor, *AAAI Spring Symposium on Planning in uncertain and changing environments*, Stanford, CA, Mar. 27–29 1990. Systems Research Center, Univ. of Maryland.

[17] R. Pelavin. *A Formal Approach to Planning With Concurrent Actions and External Events.* PhD thesis, Univ. of Rochester, Dept. of Computer SC., Rochester, NY, 1988.

[18] J. Sanborn and J. Hendler. A model of reaction for planning in dynamic environments. *AI in Engineering*, 3(2):95–102, 1988.

[19] C.J. Sellers and S.Y. Nof. Performance analysis of robotic kitting systems. *Rob. & Comp. Integ. Manuf.*, 6(1):15–24, 1989.

[20] Y. Shoham. *Reasoning about change.* MIT Press Series in Artificial Intelligence. MIT Press, 1988.

[21] X. Xiaodong and G.A. Bekey. Sroma: An adaptive scheduler for robotic assembly systems. In *IEEE Int. Conf. Robotics & Automation*, pages 1282–1287, Philadelphia, PA, 1988.