Reactive Planning

D.M. Lyons & A.J. Hendriks

Philips Research
Philips Electronics North America Corporation
345 Scarborough Road
Briarcliff Manor
NY 10510

In this article we review the field of reactive planning. Work in this relatively new field is concerned with developing techniques to produce intelligent behavior for agents operating in uncertain and dynamic environments. An uncertain environment is one in which the agent can have only limited knowledge about the environment and about the effects of its actions. A dynamic environment is one in which other entities beside the agent are active, and their interference with the actions of the agent may cause time-dependent phenomena.
The initial motivation for this new field was a dissatisfaction with the brittle nature and the computational complexity of the state of the art "classical" planning systems such as MOLGEN and SIPE when applied to 'real-world' problem domains such as robotics. The characteristics of 'real-world' problem domains are uncertainty about the effects of actions and the state of the environment, and dynamic effects caused by other active agents. In response to the deficiencies of these systems, the field of reactive planning has produced a collection of techniques, each addressing part of the general problem. However as of yet no single paradigm has emerged. In this article we group the work into three major areas: (i) Architectures for reactive machines; (ii) Design of reactive machines; and (iii) Planning for reaction. We discuss each of these areas, and conclude with an overview of what problems have yet to be addressed.

Keywords: Planning, Plan Generation, Reacting, Reaction, Plan Monitoring, Incremental Planning.

Invited contribution to the 2*nd* edition of the *Encyclopedia of Artificial Intelligence* (S. Shapiro, Editor-in-chief) to be published by John Wiley & Sons, Inc. in 1992.

1 Introduction

Planning is used in AI to refer to a body of techniques to automate the process of selecting and carrying out actions to bring an environment to some desired state. In *Classical Planning* (e.g., STRIPS (Fikes & Nilsson [13]), NONLIN (Tate [34]), MOLGEN (Stefik [33]), DEVISER (Vere [35]), SIPE (Wilkins [36]), TWEAK (Chapman [8])), an application domain is encoded in terms of a set of primitive actions and their preconditions and effects characterized as state predicates. A planning problem consists of this domain description plus an initial and final (or goal) state. Planning in this framework consists of searching through the space of action orderings to find one which takes the initial state to the goal state. This set of ordered actions is called the plan. The 'blocksworld' is the archetypical domain for classical planning: a world consisting of a number of blocks which the agent (usually thought of as a robot) can stack upon each other. The agent is the only entity active in the environment, and the configurations of the blocks are precisely known at all times.

Many real-world application domains, particularly robotic applications, do not have the static characteristics of blocksworld. Schoppers [30] describes an example domain that contrasts strongly with the blocks world: his 'baby' world. This domain is similar to the blocksworld, except it is inhabited by a "mischievous baby who will flatten block towers, snatch blocks out of the robot's hand, and even throw blocks at the robot." The crucial new ingredients in this problem domain are (1) that the agent cannot be certain of the effects of its actions, (2) it cannot make the assumption that the world remains static and unchanging except when the agent carries out an action, and (3) it cannot assume that it knows everything about the world. This is exactly the problem faced by a robotic machine operating in the 'real-world' — the same uncertain(1,3) and dynamic(2) environment that humans inhabit in everyday life.

Classical planning becomes too brittle in these application domains (Kaelbling [18], Chapman [8], Brooks [6], Agre & Chapman [1]) for two main reasons: because the world can change while planning is in progress, partial plans may be thus rendered useless; and, because of the uncertainty about the effects of actions, 'correct' plans may actually fail to achieve their goal. Chapman [8] summarized the state of the art in classical planning. He shows that classical planning, in the general case, is undecidable, and even in its simpler forms can be computationally intractable. Furthermore, Arbib [**?**] has maintained for some time that behavior should not be produced as the result of symbolic reasoning from axioms, but rather as the result of cooperation and competition between concurrent, active agents called *schema instances*. His schema theory is a bridge between Cognitive Science, Brain Theory and Artificial Intelligence, and implementations of it have been made by Lyons [21], Arkin [3] and Draper et al. [10], among others.

Recently, work in building planning systems that can operate well in application domains such as Schoppers' 'baby-world', has shown up in a number of workshops and conferences, e.g., the 1986 Timberline Lodge workshop, the 1988 Rochester Planning Workshop, the 1990 and 1991 AAAI Spring Symposia, the 1990 Workshop on Innovative Approaches to Planning, Scheduling, and Control. This work has been grouped together under the name *Reactive Planning.* It has been noted that 'reactive planning' is dangerously near a contradiction in terms, reaction usually being considered acting *without* planning. For the purposes of this article, we will continue to use "Reactive Planning" to refer to the body of techniques developed to cope with the selection and carrying out of actions to achieve objectives in an uncertain and dynamic environment. The goal of this field is to combine the advantages of a reactive system, robustness and time-critical response, with the advantages of a planning system, look-ahead and global optimality. However, it can be argued that any such integration may well lead us to redefine what we mean by both planning and reaction.

The remainder of this article is organized as follows. In the next section, we describe the characteristics of the reactive planning domain via a running example of an automated tour guide. Subsequently, we present a selection of the key reactive planning techniques and describe how they can be used to address our tour guide example. We conclude with a summary of the state of the field, and a discussion of what problems remain open.

2 Characteristics of the Reactive Planning Domain

In this section we introduce the characteristics of the reactive planning problem domain via a running example: an automated New York City tour guide. Basically, the duty of a tour guide is to take a tour group to a set of landmarks, explaining each one and answering any questions there are. However, in practise, there is more to being a good tour guide than this. First of all, a tour guide must fulfill one of the most stringent aspects of a reactive planning problem — being prepared to act and interact *at any time.*

Reactivity. Continual vigilance is essential to a reactive system. For example, a tour guide is continuously on duty once the tour starts. The tour group can ask questions about almost anything at anytime, and the guide should be prepared to respond. There is also a host of other tasks, such as making sure everybody gets back on the bus before going to the next location, working around the weather, considering the preferences of the audience, guarding the general safety of the tourists, etc. The classical planning approach involves the *a priori* complete or incremental generation of a plan, which is then sent to a plan executer to be carried out. A plan executer can do nothing without a plan to execute. A reactive system needs to be able to respond to the environment without necessary recourse to plans.

Timely Activity. Coordination with externally imposed time constraints is unavoidable in a reactive domain. For example, the tour guide has to deal with the fact that the timeliness of actions is important. Parks, museums, zoos, among others, have specific opening hours which must be considered if the tour is to remain on time, and which also constrain rescheduling if there are any disturbances to the tour. The tour can only stop for a limited time at each location. The next location needs to be given to the bus driver in advance. Time constraints are involved in reactive planning in three ways: the agent must carry out actions in a timely fashion, it must consider the effects of time in choosing which actions are appropriate, and the strategy for *choosing* the next action must also abide by time constraints. As an example of this third constraint, note that the strategy for deciding where to go next can only occupy some fraction of the time allocated to visit a location.

Uncertainty. Dealing with uncertainty is a fact of life for a reactive system. A tour guide does have a lot of *a priori* information with which to plan out the tour. Nonetheless, making a plan in the sense of the classical planning approach, *a priori* production of set of actions and some ordering information, is out of the question, since the uncertainty about the state of the environment and the actions of other entities during plan execution time (i.e., the duration of the tour) is too large: weather forecasts tend to be imperfect, and the preferences and humors of tour group members are notoriously fickle, not to mention the traffic patterns of NYC! For example, a good tour guide will visit outdoor locations, such as Central Park, when it is dry. Indoor activities will be scheduled for those times when it rains. In NYC, this will, on occasion, demand hasty improvisations. Improvising the completion of a tour will require thinking about timely coordination with entities such as museums, parks, and restaurants, as well as maintaining the on-going interaction with the audience. In such a case, it is be better to keep your audience dry and interested than to spend time deriving an optimal perturbed tour!

Improvisation and Interaction. Effective behavior in a reactive planning domain only comes about as the result of the close interaction of agent and environment. For example, for a tour guide, special events, such as parades or street theater, can be both a nuisance (since routes might have to be changed) and also an opportunity to expose tourists to those events. A good tour guide should take advantage of opportunities when possible. This may mean adding new locations to the tour or selecting different paths between locations. With a human tour guide, the description and emphasis in the talks delivered at each location would also change smoothly. As another example, a good guide asks the tourists for their preferences beforehand and during the tour, and may adapt the tour accordingly. Thus, a good tour is not a sequence of actions carried out by the guide on a passive audience, but rather a continual interaction between guide and audience. It is clearly less successful to carry out a predetermined tour and then hear the complaints afterwards and try to convince the complainers to take another tour which will incorporate their wishes (i.e., a 'recovery from error' paradigm).

Note, of course, that actually building an 'automated' tour guide would require the application of many more areas of AI in addition to reactive planning: speech understanding, computer vision, tutorial systems, cooperative problem-solving, user modeling, path planning, redundant kinematics, etc. In this example, the reactive planning component concerns the selection and carrying-out of actions appropriate to the situation.

3 Techniques for reactive planning

In this section we look at the techniques that have been developed for dealing with reactive planning problems. The first works in this area date from about 1986[1] and the field is still far from solved at present. The techniques developed in this field can be effectively classified into three groups, based on their objectives:

(i) Architectures for Reactive Machines.
A reactive machine is a system in which the choice of next action is based on hardwired response to sensory input and built-in goals. It differs from a classical 'plan executer' in that it is always ready to carry out actions; it isn't waiting for a plan to be loaded. Work in this area has the objective of developing architectures, representations and algorithms that can be used in building reactive machines that exhibit intelligent and robust behavior. This was one of the first parts of the field to be explored, and it produced the somewhat unexpected results that it is possible to build a reactive agent that produces behavior that an observer would classify as intelligent, despite the lack of classical planning and reasoning in the agent.

(ii) Design of Reactive Machines.
One of the first criticisms of reactive machines was that their robustness and intelligence really depended on the skills of the programmer who built them. This second area of reactive planning responded to this problem by designing tools for automatically building reactive systems to fulfill

[1]Though the inspiration could indeed be traced back to the STRIPS Triangle Table work from the 1970s.

a set of desired criteria. The machine generation stage is assumed to be 'off-line'; a specification of the machine and its environment is written and then input to the generation tool, which produces a detailed description of the reactive machine.

(iii) Planning for Reaction.
Another major criticism of reactive machines is that they were 'myopic' — unable to rise above local responses to the environment. This third area of reactive planning responded to that problem by trying to integrate the concept of planning-ahead with that of reaction to the environment.

The following subsections explore each of these areas in turn.

3.1 Architectures for Reactive Machines

Chapman finished his TWEAK (Chapman [8]) paper by suggesting that improvisation might be a better paradigm than planning. Agre & Chapman [1] went on to carry out the first step in the road to building systems that can exhibit intelligent behavior in uncertain and dynamic domains. They argued that "before and beneath any activity of plan following, life is a continual improvisation, a matter of deciding what to do *now* based on how the world is *now*." They built a program, *Pengi,* based on these concepts. Pengi played a video arcade game called Pengo. In a typical Pengo game, the computer appeared to hunt down targets, build traps, escape ambushes, take advantage of opportunities and act in a timely fashion. However, the intelligent behavior of Pengi was the result of the interaction of some relatively simple opportunistic strategies with a complex, structured environment. Two key ideas developed for Pengi were the concept of *routines* and the concept of *indexical-functional* representation.
Routines A routine is a pattern of interaction between agent and environment. A routine need not be explicitly represented by the agent, but can simply be a property of the regularity in the interaction between the environment and the agent. For example, tour groups exhibit certain behavioral regularity: they may listen to the guide, they ask questions, they wander off, etc. The tour-guide will typically explain some land-mark, accepting and responding to questions as they arise. This highly interactive pattern is an example of a routine. Internally, the tour-guide may simply have rules that say "answer any asked question" and "describe current landmark". The routine is created by the interleaved effects of these rules driven off the behavior of the environment (tour group). Recognizing and exploiting regularity in the environment allows for the construction of simpler and more robust agents that, nonetheless, can effectively cope with uncertainty.
Indexical-Functional Representation. In an indexical-functional representation (called *deictic representation* in the more systematic exposition in Agre (in press)[2]), properties of the immediate environment

Figure 1: The traditional architecture vs. the subsumption architecture.

are only represented in terms of the impact they have on the objectives of the agent. For instance, a tour-guide may have to deal with many people and many tours in any day. A classical planning approach would involve uniquely naming each individual encountered, e.g., PERSON-23. This introduces combinatorics, since the space of persons must now be searched for appropriate instances. A Pengi-like tour guide however would not uniquely name every person it encountered, but would only represent people in terms of their impact on its objectives, e.g., `person-now-asking-question`, or `person-in-danger-of-being-lost`. The perceptual system directly matches the environment with such indexical-functional entities, and no search is required. Pengi used a similar mechanism to directly measure from the environment clues to what actions to take next; these are called indexical-functional *aspects*. A Pengi tour-guide might have such aspects as `tour-becoming-bored` (and hence should be livened up), or `street-theater-in-view` (consider diverting to look at it), etc.

Behaviors and Subsumption. Brooks [6] made the point that the standard view of intelligent robot control architectures as a pipe-lined collection of functional modules caused problems with robustness, buildability and testability. He suggested a novel architecture, called the *subsumption* architecture, that emphasized building intelligent control from layers of task-achieving behaviors (see figure 1) much in the spirit of Braitenberg[4]. He built a number of robot system (Brooks [6, 7]) that exhibited similar behavior to Pengi: they appeared to act in a robust intelligent way in complex environments. And like Pengi, internally they consisted of simple opportunistic control rules.

Brooks builds his systems as networks of augmented fine state machine modules. Each module can accept input, has internal state, and can produce output. In addition, the output of a module can be *inhibited*, or the input *suppressed* and replaced. (Brooks associates a time interval with the inhibition

Figure 2: Levels 0 and 1 of a Subsumption Tour-Guide.

Figure 3: Levels 0, 1 and 2 of a Subsumption Tour-Guide.

and suppression, we will omit this in our simple examples here.) For example, a particularly simple tour guide could be described by two connected modules: LOCAL-WANDER, which produces random heading signals, connected to a MOVE which when given a heading will move in that direction. A subsumption architecture is a hierarchy of such behaviors, high-level ones subsuming the behaviors of lower-levels but adding in some extra behavior where appropriate. If we consider the 'wander-move' behavior as a level zero of a subsumption architecture, we can build a level one architecture that 'directs' the lower level when appropriate, but otherwise remains silent (see figure 2).

Level 1 consists of a counter (UP COUNT) that indexes through the list of landmarks, and for each one, suppresses the local wander behavior to force the guide to that location, and then triggers a speech at that location. While the guide is at a location, the local wander behavior becomes unsuppressed and the guide wanders around the landmark. This is a more useful tour-guide, but it is still rather

Figure 4: Nilsson's Action Unit.

inflexible. We can add a third level that allows for reorganization of the schedule should the weather turn nasty, e.g., figure 3. If bad weather is seen by the WEATHER module, it inhibits the counter output and the signal is replaced with a succession of headings generated by the INDOOR-SITES modules — headings for landmarks that are known to be indoors.

<u>Goals and Beliefs.</u> Nilsson [24] has put forward an action network structure for teleo-active agents, agents that take changing environmental and goal conditions into account in acting. A strong motivation for this work was the dissatisfaction with conventional action control methods, where higher-level units surrender control when they activate a lower-level unit and have to suspend until control is explicitly returned. This scheme can cause low-level units to continue to operate in situations for which they are no longer relevant, since changes in situation go unnoticed by the suspended higher levels. To overcome this, Nilsson proposed a new scheme for organizing behavior based on networks of combinatorial circuits in which higher-level routines enable lower-level ones but do *not* surrender control. In addition, he developed a language, ACTNET, that allows a user to specify these control networks as networks of combinatorial circuits. The goals and beliefs of an agent are explicitly represented in this network. The planning component of a network is 'built-in' in the sense that goals are hard-wired to fire particular actions.

The basic unit in Nilsson's network is an action unit (see figure 4). An action unit is a logical gate that decides whether to fire a particular action based on the goal request G, the absence of goal establishment (the purpose P) and the preconditions P_i of the action. An action unit continually monitors its input, and fires the action as soon as it is applicable and executable. Goals, as well as preconditions, can be dynamic in nature ('talk so loud that all the tourists can hear it' is dependent

Figure 5: Part of a Tour-Guide Action Network.

on the background noise level). Networks are constructed by tying together the signals representing the goals, preconditions, etc., in an appropriate fashion. Higher-level networks can enable lower-level ones by turning on their purpose input.

As an example of an network, consider the case of a tour guide trying to show the Empire State Building(ESB) to his group (see figure 5). This goal fires two primitive goals, i.e., `at-ESB` and `present-ESB-speech` via action unit `Show-ESB`. These establish the goals for the units `move-to-ESB` and `present-ESB-speech`. Since `present-ESB-speech` has as its precondition `at-ESB`, it has to wait until the other action unit `move-to-ESB` has established that precondition.

In later work Nilsson [25], generalizes the notion of action networks to structures that reactively execute sequences of goal-seeking actions, and develops a language for programming such structures. Whereas action units achieve their goals single-handedly, *teleo-reactive* structures eventually achieve their effects by conditionally enlisting other teleo-reactive structures or actions to achieve necessary subgoals.

Monitoring change. In dynamic worlds, change proceeds regardless of the agent's state. Therefore, keeping track of the state of affairs is a non-trivial task in such domains. Hendler's DR (Hendler [15], Sanborn & Hendler [29]) is an approach to managing observation and actions in dynamic domains. In DR, monitors are employed to keep track of the situation in the environment. A monitor is an *independent* information gathering component of the overall reasoning system that maintains the world model. The duty of a monitor is to report significant events.

Monitoring is coupled to reaction via constraints reporting. Every action has a set of constraints on its execution state. Constraints are divided into inhibitory constraints, constraints that prevent an applicable action from executing, and enabling constraints, constraints that allow an applicable action to execute. Action selection is determined on basis of the active constraints. If an enabling constraint is found on an action, then that action is performed. If an inhibitory constraint is found on an action, then the action is suspended and another applicable action is pursued until the constraint is removed.

3.2 Design of Reactive Machines

Reactive machines, machines which produce intelligent, but improvised, behavior, were a major step forward in addressing the problem of acting in uncertain and dynamic environments. The advantage of such machines is that they respond quickly and robustly, and can be surprisingly intelligent given their lack of internal models. However, much of the power of a Brooks-style robot, for example, comes from the skill with which the behaviors have been designed and composed together in the subsumption architecture. A major direction of work in reactive planning tries to remove this manual element by developing techniques to design formally correct machines, and by providing tools for automatically generating a reactive machine to suit a set of design criteria.

Universal Plans: the autogeneration of reactive machines. The Universal Plan work (Schoppers [31]) is an approach to building reactive machines as highly conditional plans. A universal plan is a plan that can achieve a goal given any possible initial state of the environment. In this approach, actions are selected through the classification of the actual situation encountered at execution time, as opposed to the classical approach of selecting actions based on a single initial situation at planning time. Thus in a universal plan, the failure of an action to achieve its predicted effects does not necessitate replanning, only the selection of a new initial point from which to execute the plan. (See figure 6). The task of a planner has, therefore, changed from generating a sequence of actions, to anticipating all possible situations and concomitant appropriate responses. At execution time, a sensing module recognizes the

Figure 6: The Classical (a) vs. the Universal (b) Plan Approach.

actual situation and selects the appropriate action to perform.

For example, a (simplified) universal plan for a tour guide who is showing the Empire State Building (ESB) is shown in figure 7. The goal, to have the landmark shown to the tour, initiates the two subgoals `at-ESB` and `ESB-speech` (by the action `SHOW-ESB`). Carrying out the universal plan consists of examining the the truth values of these two goals, and if they are not both true, then an appropriate universal plan is employed to achieve each goal. For instance the universal plan for `at-ESB` is to move to the location (`MOVE-TO-ESB`), but only on the condition that the group is already complete. If this condition does not hold, then the group must first be assembled. Notice that the failure of the action `MOVE-TO-ESB`, for any reason, results in the predicate `at-ESB` remaining false. This allows the action to be attempted again without the need for replanning. Once the group is at the location, the guide can then pursue the other goal of actually presenting the history of the building.

Situated Automata: Formally correct reactive machines. The *situated automata* model of Rosenschein & Kaelbling [19, 27, 28] has influenced many workers in the field of reactive planning. This model was developed to address the issues of real-time performance and provably correct behavior. A program in this model is a finite machine with internal state that transforms inputs to outputs within a small time period. This model, therefore, provides a good base on which to built reactive systems. A Lisp-like language, called Rex, was developed to specify such machines. The output of a Rex program is a description of a digital circuit for a machine that meets the Rex specification.

A key characteristic of the situated automata model is that the correctness of the machine's potential

Figure 7: A Universal Tour-Guide.

behavior is addressed by considering the *knowledge* that the machine contains, where knowledge is analyzed in terms of the relationship between a machine and its environment. A machine is defined to know a proposition ϕ in machine state s, if whenever the machine is in state s, ϕ is satisfied. Given a Rex program, a background theory describing facts about the environment, and a description of the epistemic meaning of the machine inputs, the situated automata model provides a methodology for attacking the problem of verifying the meaning of the machine outputs.

Kaelbling [19] describes a compiler, Gapps, that maps a goal specification and a set of goal reduction rules to a Rex program for a machine to achieve those goals. The goal reduction rules capture domain specific information. In this fashion, a correct reactive system can be automatically generated off-line.

3.3 Planning for Reaction

A crucial problem with any one reactive machine is that should the environment diverge from that in which the machine was designed to operate, then the machine may produce inappropriate behavior. The best a reactive machine can do in the face of such a change is to degrade gracefully; that is, its responses may not be as efficient or appropriate in the new environment, however, these responses will not result in self-destruction. To address this problem completely, it is necessary to integrate the concept of the reactive machine with the concept of *a priori* planning of some kind. Note that these two concepts have some complementary characteristics: a reactive machine is not be able to deal with the need to look ahead and plan, while an on-line, time-constrained planner could not react sufficiently quickly to deal with unexpected hazards. In this section we look at some approaches to the problem of integrating a planning component with a reactive machine.

<u>Time-constraints on planning.</u> Interaction with a dynamic world forces time-constraints onto both plan-

Figure 8: Hendler's Planning-Reaction Hierarchy.

ning and reaction. Reactive systems can usually cope with these constraints, however, it is more difficult for planners. Dean & Boddy [9] introduced the class of "anytime" algorithms to deal with time-dependent planning. An algorithm has the anytime property if it can handle preemption, will return a sensible answer at any time when terminated and, most importantly, will return answers which improve monotonically over time. The design of anytime algorithms plays a key role in many approaches to reactive planning.

Integrating planning and reaction in a hierarchy. Hendler [16] notes the conflicting needs of reactors and planners: the former need input that closely matches the external world and need to respond on a very short time-scale, the latter need to abstract away from the details of the external world and may need to work for some time on a problem. The solution suggested is a hierarchy as shown in figure 8, based on the DR model discussed earlier. A scheduler chooses planner or reaction agents on a time permitting basis.

To investigate this scheduling approach, Spector and Hendler [32] propose a model in which planning and reaction interact across five different levels: sensory/motor, spatial, temporal, causal and conventional (i.e., general world knowledge). This multilevel reasoning is realized in a parallel, blackboard-based planner called APE (Abstraction Partitioned Evaluator). As an example of how APE would reason, suppose that the tour guide has a plan for conducting a tour that includes visiting the Empire State Building. Upon arrival however, the tour guide sees an excessive line for entrance to the Empire State Building. This would first be reported as a problem at the spatial level. Replanning at the that level is then immediately attempted (e.g., find another, shorter line), but if that fails, then the problem is passed to the next highest level for solution, and so on. During higher level reasoning,

Figure 9: ERE Reductor-Projector-Reactor Architecture.

the tour guide would be using more of the resource on those levels, and would be less able to react to change in the environment, thus the propagation of information to higher levels only occurs when time (and the enviroment) permits.

Planning to guide reaction. Bresina & Drummond [5] introduce an alternate approach to systems that have to produce intelligent action under time constraints. Their domain of application is photoelectric telescope scheduling problems. They suggest an architecture, called ERE, that combines the ability to react to the current environment with the ability to plan ahead. Their system consists of three independent and concurrent components (see figure 9): a *Reactor*, based on Drummonds plan-net formalism, that reacts to the current environment; a *Projector*, that determines the future effect of possible next actions, and advises the Reactor on which ones best satisfy the system's objectives; and a *Reductor* that advises the Projector about which possible next actions are the appropriate ones to explore given the systems objectives (i.e., the Reductor provides search control to the Projector).

The ERE architecture evolved from Drummond's work on the plan-net formalism (Drummond [11]). This formalism is based on net theory (Peterson [26]), and was designed to provide the ability to represent iterated actions and the sensory effects of actions. For example, the fact that a tour guide can deliver a speech about some landmark only if he is at that landmark and he hasn't delivered the speech before on that tour is represented by the plan net of figure 10. Circles represent conditions that must hold. Boxes represent events. When arcs join conditions and events, this denotes that the conditions enable the events. When arcs join events to conditions, this denotes that the events cause the conditions. There is much more to plan-nets than this condition-event theory. However, this is sufficient for our examples. The net in figure 10 demands that the tour guide be at a landmark,

Figure 10: Example Plan-Net.

x, at which a speech has not previously been given, $\neg done(x)$, before the action event `give-speech` can occur. Once the action occurs, the following effects are asserted: a physical change to the world occurs because the speech has happened, and the internal representation of the tour-guide is updated to include $done(x)$.

An ERE Reactor component for a tour guide that presents landmarks at random is given by the set of one instance of each of the three nets in figure 11 for each landmark location $x \in Landmarks$. The first net (a) is simply the speech net of figure 10. The second net 'motivates' the tour guide to visit new landmarks. The third net models the fact that a tour will get wet if it visits an outdoor landmark while its raining. In any given situation s, the Reactor derives the set of actions that can be carried out by the tour-guide. These are simply the set of actions that are enabled in the plan-net by the situation s, $enabled(s)$. One of these is chosen non-deterministically and carried out.

This Reactor is capable of exhibiting all tour behaviors. However, it does so in a myopic fashion, since its only criterion for selecting actions is that they be enabled. The objective of the Projector component is to use a description of the Reactor's plan-net to determine, in this case, which tours are acceptable given the current environment and a set of desired behavor constraints.

For example, consider an automated tour guide which has explicit instructions not to get the tour wet. For all worlds in which it doesn't rain, the Reactor is capable of fulfilling this constraint. However, should the world change, and it start to rain, then the Reactor, on its own, will not obey this constraint. This constraint, expressed in ERE's behavioral constraint language would be `(prevent tour-is-wet tour-start-time tour-end-time)`. Using estimates of the probability of conditions and the utility of actions, the Projector component projects forward in time the effects of action sequences that could be selected from the Reactor's plan-net (Drummond and Bresina [12] describes a theory of temporal projection based on these plan-nets). When it finds the first sequence that obeys the behavioral constraint, it compiles a set of *Situated Control Rules* (SCRs) and sends them to the

Figure 11: ERE Tour-Guide.

Reactor. These are if-then rules that offer the Reactor advice on the best subset of $enabled(s)$ to carry out to keep in line with the behavioral constraint. In our example, the SCRs would advise the next acceptable place to go, i.e., that subset of $enabled(s)$ where $indoors(s)$ is true. Once the Projector has found one good sequence, it continues by inspecting other less likely conditions and again updating the Reactor with SCRs when available. The nature of Projector-Reactor interaction is as follows: Without any planning, the Reactor should be able to give some sort of a tour; with some planning, the Reactor should be able to give, say, a single acceptable tour; and with lots of planning, the Reactor should be able to give many different acceptable tours (one tour being selected over another depending on various environmental circumstances).

Planner and Reactor as a coupled system. Lyons & Hendriks et al. [22, 23] address the problem of planning for reaction by considering a Planner and a Reactor as two elements of a concurrent, cooperating system. The Planner interacts with, and is influenced by, the Reactor, in the same fashion that the Reactor interacts with, and is influenced by, the world. This architecture is shown in figure 12. Their Reactor component is based on a special purpose model of computation developed for representing highly conditional robot plans. The model, called $\mathcal{RS}$, is an extension of the *Robot Schemas* model of Lyons & Arbib [21]. This model inherits the philosophy of Arbib's schema theory (Arbib, this volume) in seeing behavior as the result of the cooperation and competition of a set of interacting schema instances. The model is process-based, and process-algebra methods[17] are used to analyze the process network behaviors. Processes can be defined in terms of networks of other processes, grounding out with a set of atomic, pre-defined, processes. This provides a powerful mechanism for specifying flexible, hierarchical structures. They make the point that to analyze the behavior of reactive machines

Figure 12: Planner-Reactor-World System.

it is very necessary to know in what sort of environment the machine will be situated. Thus, in reactive machine analysis, the $\mathcal{RS}$ model is used to represent both the plan (or controller) and the environment in which the plan will be carried out [20].

An $\mathcal{RS}$ Reactor is specified by a set of recursive process equations, where processes can be coupled in networks in two ways: they can communicate messages to each other via *communication ports*, or they can be composed together using several kinds of *process composition operators.* An $\mathcal{RS}$ tour guide reactor, **TourGuide,** that tours landmarks in some arbitrary order is illustrated in figure 13, and defined algebraically as follows:

$$\begin{aligned}
\texttt{Traverse}_{S,l} &= (\ \texttt{Done?}_l\ ,\ \texttt{Random}_S\langle nl\rangle\ )\ :\ (\ \texttt{MoveTo}_{nl}\ ;\ \texttt{Traverse}_{S-nl,nl}\ )\\
\texttt{Present}_S &= \texttt{At}_S\langle l\rangle\ :\ (\ (\texttt{Dospeech}_l, \texttt{AnswerQ})\ ;\ \texttt{Done}_l\ ;\ \texttt{Present}_{S-l}\ )\\
\texttt{TourGuide} &= \texttt{NewTour}\ :\ (\texttt{Traverse}_{L,l0}, \texttt{Present}_L)\ ;\ \texttt{TourGuide}
\end{aligned}$$

The **Traverse** network consists of a concurrent network that selects the next landmark to visit, nl, from the set of landmarks, S, while monitoring for when the current landmark has been completed; it then signals a move to the new position, after which it recurses. The **Present** network consists of a monitor process that reports when the tour guide is first at landmark l and then concurrently interleaves delivering the landmark speech, $\texttt{Dospeech}_l$ and question answering, **AnswerQ.** When these are finished it signals the landmark is done, and recurses. (The interaction between **Done?** and **Done** is a typical application of $\mathcal{RS}$ message passing.) The reactor network, **TourGuide,** recursively activates a concurrent combination of **Traverse** and **Present** for each new tour. Recursion of **Traverse** and **Present** is bounded using the conditional composition operator[2] (denoted by ':').

The Planner 'tunes' the Reactor so that the concurrent composition of Reactor and the world model,

[2]Conditional composition is special kind of sequential composition in which the second process is only created if and when the first process terminates successfully; if it aborts, then the composition aborts. If **Random** and **At** abort when given an empty argument, then the above reactor recursion is bounded.

Figure 13: $\mathcal{RS}$ Tour-Guide Reactor.

`World,` continue to produce appropriate behavior. The full system is then a concurrent composition of the Planner and the Reactor-World system: `(Planner,(Reactor,World)).` By considering a Reactor to be a set of concurrent situation-labeled actions, Lyons & Hendriks define formally what it means for the Planner to *improve* the behavior of the Reactor over time. The Planner cycles continuously, making changes to the Reactor in accordance with the definition of Reactor improvement. The Planner does not have access to the complete internals of the Reactor; its observations are restricted to the output of *perception processes* that it can embed into the Reactor. Thus, the Planner can ignore the bulk of the sensing carried out by the Reactor, and concentrate on a few, important pieces of information. Perception processes are used to 1) determine when some part of the Reactor is in danger of failing, 2) to reflect on-going resource usage, and 3) to determine whether certain goals or subgoals have been met.

For example, if during a tour, clouds start moving in, a perception process may signal this to the planner. From this information, the Planner deduces the possibility for rain and therefore the possibility of the tour getting wet if it is at an outdoor location. Therefore, it reasons to adapt the Reactor by changing the `Traverse` network to restrict the selection of landmarks to only indoor ones when it is raining, while retaining the full choice otherwise. After sending down this adaptation to the

Reactor, the latter has become self-sufficient to deal with rain or sun shine, and the Planner doesn't need to worry about weather changes again. In this fashion, the Planner can use the Reactor to focus its reasoning, and in turn the Reactor is guided by the Planner to improve its behavior.

4 Discussion and Remaining Problems

Reactive planning is still a young field. It has established that there are alternatives to classical planning when operating in an uncertain and dynamic environment; however, no single reactive-planning paradigm has emerged yet. What has emerged is a number of themes that will be important to this field.

1. Reactive machines can behave surprisingly intelligently and robustly. Reactive machines work best when they can be designed to exploit an environment, resulting in what Agre & Chapman call routines – patterns of behavior that are goal directed, but which aren't represented explicitly in the agent, rather they are brought about as a consequence of the interaction between agent and environment. The terms *situated activity* and *emergent behavior* refer to similar phenomena.

2. A number of concepts have been developed to help build reactive machines: indexical-functional representation, behaviors and subsumption, decentralized, concurrent decision making, and formal analysis of agent and world as a coupled system.

3. The main critique on the approach of achieving reactive behavior by the *a priori* design of reactive machines (as voiced by Ginsberg [14]) is that of the enormous computational complexity, both at design time (as the machine should have an appropriate action for every possible situation) and execution time (as the sensing system must be able to identify all aspects of the world that distinguish situations). The latter, however, can be partially remedied by a decision-tree approach to the determination of the actual situation in which the agent finds itself (as Schoppers notes). From an efficiency viewpoint, however, there is a problem in that large parts of the plan are irrelevant (as most situations are very unlikely to occur), yet all achieve equal attention during the design phase.

4. The weaknesses of reactive machines are clear: they can be very myopic and there are limits to their robustness. Some form of additional, longer-term reasoning is necessary, though again, it is an open question as to how to integrate reactive machines and longer-term reasoning. The importance of having planning and reaction as concurrent, interacting activities has become evident.

There are still a lot of open questions in this area, and of course, much work in other fields of planning, non-monotonic reasoning, robotics, etc., also impacts reactive planning. We list here just a few of the open areas:

1. There has been little work in integrating the reactive machine work with the extensive real-time computation field.

2. Resources play a key role in many reactive domains – the concept of 'making do' with what is available — but as yet there is no unique reactive viewpoint on resources.

3. There are a number of suggestions, but no definitive answers yet, on an appropriate way to view long-term 'planning' so as to integrate it with reaction.

4. Learning can play a key role in increasing the scope in which a reactive planning system can operate. There appears to be a synergy between reaction and learning (Sutton [**?**], Mitchell [**?**]). For example, the learning component of Sutton's Dyna occupies a similar place in his architecture to the planning component in Bresina & Drummond's ERE.

Acknowledgements

A synopsis of the diverse work in this field would not have been possible without the cooperation of the other researchers. We owe thanks to the following people for their insightful comments and helpful discussions: Phil Agre, Michael Arbib, Marc Drummond, Jim Hendler, Nils Nilsson and Marcel Schoppers.

References

[1] P. Agre and D. Chapman. Pengi: an implementation of a theory of action. In *Proc. AAAI-87*, pages 268–272, Seattle, WA, Oct. 1987.

[2] Philip E. Agre. *The Dynamic Structure of Everyday Life.* Cambridge University Press, Cambridge, (in press).

[3] R. C. Arkin. Motor schema-based mobile robot navigation. *Int. J. Rob. Res.*, 8(4):92–112, Aug 1989.

[4] Valentino Braitenberg. *Vehicles.* MIT Press, Cambridge, Mass. and London, Eng., 1984.

[5] 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.

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

[7] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47(1–3):139–160, Jan. 1991.

[8] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[9] T. Dean and M. Boddy. An analysis of time-dependent planning. In *IJCAI-87*, pages 49–54, 1987.

[10] B.A. Draper, Collins R.T., J. Brolio, A.R. Hanson, and E.M. Riseman. The schema system. *Int. J. Comp. Vis.*, (2):209–250, 1989.

[11] M. Drummond. A representation of action and belief for automatic planning systems. In *Workshop on Planning& Reasoning about Action*, pages 267–289, Timberline OR, 1986.

[12] M. Drummond and J. Bresina. Anytime synthetic projection: maximizing the probability of goal satisfaction. In *AAAI-90*, pages 138–144, Jul. 29th – Aug. 3rd 1990.

[13] 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.

[14] M. Ginsberg. Universal planning: An (almost) universally bad idea. *AI Magazine*, pages 40–44, Winter 1989.

[15] J. Hendler. Real-time reaction for planning systems. In *AAAI Spring Symposium on Planning and Search*, pages 24–26, March 1989.

[16] J. Hendler. Abstraction 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.

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

[18] L.P. Kaelbling. An architecture for intelligent reactive systems. In *Workshop on Planning & Reasoning about Action*, pages 235–250, Timberline OR, 1986.

[19] L.P. Kaelbling. Goals as parallel program specifications. In *AAAI-88*, pages 60–65, 1988.

[20] 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.

[21] 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.

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

[23] 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.

[24] N.J. Nilsson. Action networks. In J. Weber, J. Tenenberg, and J. Allen, editors, *From Formal Systems to Practical Systems*, pages 21–52. Dept. of Computer Science, Univ. of Rochester, Oct. 27–29th 1988.

[25] Nilsson N.J. Toward agent programs with circuit semantics. *Artificial Intelligence (in press)*, 1991.

[26] J.L. Peterson. *Petri-Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.

[27] S. Rosenschein. Synthesizing information-tracking automata from environmental descriptions. In R. Brachman, H. Levesque, and R. Reiter, editors, *1st Int. Conf. on Princ. of Knowledge Rep. and Reasoning*, pages 386–393, Toronto, Canada, May 1989. Morgan Kaufman Inc. CA.

[28] S.J. Rosenschein and L.P. Kaelbling. The synthesis of digital machines with provable epistemic properties. Technical Note 412, SRI International, Menlo Park, CA, April 1987.

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

[30] M. Schoppers. *Representation and Automatic Synthesis of Reaction Plans.* Technical Report UIUCDCS-R-89-1546 (PhD Thesis), Dept of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[31] M.J. Schoppers. Universal plans for reactive robots in unpredictable environments. In *IJCAI-87*, pages 1039–1046, 1987.

[32] L. Spector and J. Hendler. An abstraction-partitioned model for reactive planning. In *Fifth Rocky Mountain Conference on AI*, Las Cruces, NM, June 1990.

[33] M. Stefik. Planning with constraints (molgen: Part 1); and planning and meta-planning (molgen: Part 2). *Artificial Intelligence*, (6):111–170, 1981.

[34] Austin Tate. Generating project networks. *IJCAI-5*, page 888 to 893, 1977.

[35] S. Vere. Planning in time: Windows and durations for activities and goals. *IEEE PAMI*, 5(3):246–267, 1983.

[36] D.E. Wilkins. Domain-independent planning: Representation and plan generation. *Artificial Intelligence*, 22(3):269–301, 1984.