Testing Incremental Adaptation

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

Abstract

A robot system operating in an environment in which there is uncertainty and change needs to combine the ability to react with the ability to plan ahead. In a previous paper we proposed a solution to the problems of integrating planning and reaction: cast planning as *adaptation* of a reactive system. In this paper, we present our first experimental results from the planner-reactor architecture, and compare the approach with other work in the planning and learning literatures.

Introduction

The importance of integrating deliberative ("planning") capabilities and reactive capabilities when building robust, 'real-world' robot systems is becoming widely accepted [Bresina & Drummond 1990, McDermott 1991]. In [Lyons & Hendriks 1992a] we proposed a solution to this integration: cast planning as *adaptation* of a reactive system. Here we present our first performance results, using our approach to construct a robot kitting workcell.

We begin with a brief review of our application domain and the necessity for developing our approach. We then present summaries of the planner-reactor architecture we have developed to implement of our approach, and the kitting robot application domain. Subsequent sections provide more details of our implementation and present our experimental results. We conclude with a comparison of our approach to similar work in the planning and learning communities.

Background

Our application domain is the kitting robot: a robot system that puts together assembly kits — trays containing all the necessary parts from which to build a specific product. We consider this application to be exemplary of domains that demand the integration of reaction and deliberation. A reactive controller will allow the kitting robot to handle the uncertainty in the quality and quantity of the raw materials stream and downstream automation. However, a deliberative component is still necessary to 'program' the robot to handle different kits/parts or mixes of kits, as well as to implement temporary strategies in the face of disturbances on the factory floor.

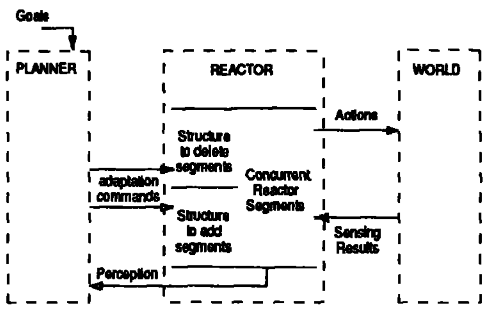
Goals
PLANNER
REACTOR
WORLD
Actions
Structure to delete segments
adaptation commands
Concurrent Reactor Segments
Structure to add segments
Sensing Results
Perception

Figure 1: The Planner-Reactor Architecture.

Some approaches to this class of integration problem have been suggested. Schoppers[Schoppers 1989] amongst others has proposed an 'off-line' generator for reactive systems. This approach works well only when the deliberative component can be separated 'off-line'. We address the problems that arise when both reactive and deliberative components need to be 'on-line'. Connell's SSS [Connell 1992] addresses this integration for mobile robots. In SSS, though, the deliberative component is restricted to enabling/disabling behaviors, while we need to be able also to generate new behaviors. Bresina & Drummond's ERE [Bresina & Drummond 1990] comes closest to our approach. A key difference is that our domain, kitting, is a *repetitive* activity. Our concept of the improvement of a reactive system exploits this repetition, whereas ERE was designed for once-off activities.

The Planner-Reactor Architecture

A *planner-reactor* system is composed of two concurrent modules, a *planner* module, which is the deliberative component of the system, and a *reactor* module, which is the reactive action execution component of the system (figure 1). The reactor contains a network of reactions: hard-wired compositions of sensor and motor processes. The key property of the reactor is that it can produce action at *any* time. Unlike a plan executor or a hierarchical system of planner and reactive system, our reactor acts ***asynchronously*** and ***independently*** of the planner. It is always actively inspecting the world, and will act should one of its reactions be triggered. A reactor should produce timely, useful behavior even without a planner.

Rather than viewing the planner as a higher-level system that loads plans into an executor, we see the planner as an equal-level system that continually tunes the reactor to produce appropriate behavior. The interaction between the planner and reactor is entirely ***asynchronous.*** The planner continually determines if the reactor's responses to the current environment would indeed achieve the desired goals. If not, then the planner makes an incremental change to the reactor configuration to bring the reactor's behavior more into line with the objectives.

In [Lyons & Hendriks 1992b] we formally defined the reactor (using the $\mathcal{RS}$ language) as a set of concurrent reactions and a well-defined interface for adding to or deleting from that set. Such addition or deletion of reactions we call ***adaptation.*** In [Lyons & Hendriks 1993] this formal treatement was extended to handle ***safe*** adaptation, i.e. adaptation that does not interrupt reactions, that is guaranteed to finish, and that will not produce intermediate behaviors.

In [Lyons & Hendriks 1992b] we also presented the theory behind incremental improvement of the reactor. We introduced the concept of the ***ω-ideal reactor***, a reactive system that operates effectively as long as the assumptions in the set ω hold in the environment. We proposed the following incremental reactor construction strategy: ω is initially chosen to allow the planner to quickly produce a working reactor and then ω is gradually relaxed over time. In addition to this 'normal' relaxation sequence, the planner can also be ***forced*** to relax assumptions. Whenever the planner builds a reactor that depends upon a particular assumption holding, it needs to embed a monitor process that can detect whether the assumption actually holds in practice or not. The monitor process triggers a 'clean up' action if the assumption ever fails, and notifies the planner that it needs to relax this assumption next.

The Kitting Robot Problem

A robot kitting scenario was used in [Lyons & Hendriks 1992b, Lyons & Hendriks 1993] to explain the planner-reactor approach. We continue the example in this paper to explore the performance of a planner-reactor system in practise.

The Kitting Task. Assembly components are fed from stores to a kitting station on a conveyor belt. The kitting station consists of one or more kitting robots. Each kitting robot must take parts off the belt, place them in the appropriate slots of a kitting tray, and then place the kitted tray onto the belt. The trays are stored in a stack in the robot's workspace. The product we are kitting is a small DC servomotor manufactured by Philips. It has three parts: a cap, a motor and a body, all of which have a number of variants.

Kitting Assumptions. The planner-reactor approach provides a way to incrementally construct reactive systems with improving performance. At the heart of this iterative mechanism is the concept of characterizing the environment by a set of assumptions. The following are four of the (nine in total) assumptions developed for the kitting application. Some of the assumptions, such as that of the quality and substitutability of parts, ***must*** eventually be relaxed to get to a robust workcell. Some other assumptions describe contingencies that make a more versatile workcell, but are not generally necessary for a robust system. Still other assumptions pertain to unusual operating conditions, such as the assumption of no downstream disturbances.

1. **Assumption of Parts Quality (AQ): All the parts coming into the kitting workcell are of good quality and do not need to be tested.**
2. **Assumption of non-substitutability of parts (AS): Each part has only one variant.**
3. **Assumption of no parts motion (AM): The kit parts do not move around once delivered on the belt to the workcell.**
4. **Assumption of no downstream disturbance (ADD): Downstream automation is always ready to receive finished kits.**

The $\mathcal{RS}$ Model

We employ a language called $\mathcal{RS}$ to represent and analyse the planner-reactor framework. $\mathcal{RS}$ [Lyons 1993] was developed to represent and analyze the kind of programs involved in sensory-based robotics. This language allows a user to construct programs by 'gluing' together processes in various ways. This style of language is called a ***process-algebra*** language. The language is described extensively in [Lyons 1993] and here we include only a short introduction.

In $\mathcal{RS}$ notation, $\mathtt{P}_m\langle x\rangle$ denotes a process that is an instance of the schema P with one ingoing parameter m and one outcoming result x. Process networks are built by composing processes together using several kinds of *process composition operators.* This allows processes to be ordered in various ways, including concurrent, conditional and iterative orderings. At the bottom of this hierarchy, every network must be composed from a set of atomic, pre-defined processes. The composition operations are:

- ***sequential*** **A;B**, do **A** then **B**.
- ***concurrent*** **A|B**, do **A** and **B** concurrently.
- ***conditional*** **A:B**, do **B** only if **A** succeeds.
- ***negation*** **~A**, do **A** but fail if **A** succeeds and vice-versa.
- ***disabling*** **A#B**, do **A** and **B** concurrently until one terminates.

We use two recurrent operators:

- ***synchronous recurrent*** **A :;B** $\triangleq$ **A : (B;(A :;B))** while **A** succeeds do **B**.
- ***asynchronous recurrent*** **A :: B** $\triangleq$ **A : (B | (A :: B))** while **A** succeeds spawn off **B**.

Situations

Reactor situations are a mechanism to group related reactions together in a hierarchical fashion. This modularity is important because it allows (human) designers

- **$\mathbf{ASSERT}_{s,p,...}$ Assert situation s and initialize the situation parameters to $p, \ldots$. This terminates when the situation is terminated and stops or aborts depending on whether FAIL or SUCCEED was used.**
- **$\mathbf{FAIL}_{s,k}$ Terminates instance k of situation s with fail status.**
- **$\mathbf{SUCCEED}_{s,k}$ Terminates instance k of situation s with success status.**
- **$\mathbf{SIT}_s\langle k, p1, p2, \ldots\rangle$ Terminates if an instance of situation s is currently asserted. k is the instance number and $p1, p2, \ldots$ are the current values of the situation parameters.**
- **$\mathbf{REASSERT}_{s,p,...}$ Assert s and whenever it ends with fail status reassert it.**

Table 1: Situation basic processes.

to more easily understand the reactor and diagnose any problems that may occur; it provides the planner with a unit around which to define safe changes to the reactor structure [Lyons & Hendriks 1993]; and it provides a way to give computational resources to those reactions that currently need it, while 'suspending' others.

Intuitively: a situation being active expresses the appropriateness for the reactor to enable the set of reactions associated with that situation. Situations can be nested hierarchically, and many situations may be active at one time so that the execution of their reactions will be interleaved. We introduce a small set of basic processes with which to build reactor situations in $\mathcal{RS}$ (table 1). For example, an instance of a situation is asserted by the execution of the $\mathbf{ASSERT}_{s,p1,p2,...}$ process, where $p1, p2, \ldots$ are the values for the parameters associated with the situation. The $\mathbf{SIT}_s\langle k, p1, \ldots\rangle$ process, when executed, suspends itself until an instance of situation s is asserted. It then terminates and passes on the details of the situation instance as its results.

We use the $\mathbf{SIT}_s$ process to ensure that a reaction associated with a situation is only 'enabled' when an instance of that situation is asserted. For example, let $\mathtt{P}^1, \ldots, \mathtt{P}^n$ be the reactions for situation s, then we would represent this in the reactor as

$$\mathtt{PS} = \mathtt{SIT}_s :; \mathtt{P}^1 \mid \mathtt{SIT}_s :; \mathtt{P}^2 \mid \ldots \mid \mathtt{SIT}_s :; \mathtt{P}^n$$

The ':;' operation ensures that as long as the situation is active, then the reactions are continually re-enabled. Note that once enabled, a reaction cannot be disabled until it has terminated. The asserting and termination of instances of situation s happen as the side effects of $\mathbf{ASSERT}_s$ and $\mathbf{FAIL}_s$ or $\mathbf{SUCCEED}_s$ processes in reactions in other situations.

Planning as Adaptation

This section outlines the principles of operation of the planner, a detailed description of its architecture is given in [Lyons & Hendriks 1992b].

Adaptation Increments. At every planner iteration t, the planner generates what we call an *expectation*, E_t; an abstract description of the changes it expects to make to the reactor to achieve the current goals G_t. The combination of the reactor model R_t and the expectation is always the ω-ideal reactor (where ω is the current set of assumptions the planner is working with).

To reduce this expectation, the planner reasons within a Problem Solving Context (PSC) consisting of the relevant parts of the environment model, the action repertoire and the assumptions ω. The outcome will be a reactor adaptation ΔR_t, the reactions necessary to implement the expectation reduction ΔE_t. Given a set of goals, the initial expectation reduction will be the construction of an an abstract plan. This abstract plan is incrementally transferred into the reactor by the insertion of situations, and is also used for search control in the planner itself. This has two advantages: firstly it allows the planner to adapt the reactor partially, with some of the reactor segments being a **STUB**, i.e., a process that does nothing except notify the planner that this particular segment has become active. Secondly it allows modular refinements due to, e.g., assumption relaxation later on.

For the kitting robot domain, an initial abstract plan would be a sequential composition of the situations *AcquireTray*, *FillTray*, and *RemoveTray*. Only *AcquireTray* needs to be concrete in order for the planner to adapt the reactor. The other situations can have **STUB** reactions initially. The planner can refine these while the reactor acquires a tray.

The process descriptions below show a reactor segment generated by the planner for this initial adaptation. Each reaction in the segment below is in an if-then-else form (`~(Cond : IfTrue) : Else`). The **REASSERT** process continually reasserts failing situations. **P0** is necessary to assert the topmost situation repeatedly. The process $\mathbf{Move}_{obj,loc}$ repositions a grasped object obj to a location loc.

Barring any perceptions received from the reactor, the planner will continue to incrementally flesh out abstract segments and adapt the reactor, until all the reactor segments are made concrete, i.e., all the **STUB** processes removed. The planner has then (by definition) achieved an ω-ideal reactor, and can proceed to relax the next assumption. If any segment relies on an assumption then the adaptation will additionally contain an assumption monitor for that assumption.

$$\begin{aligned}
\mathtt{P0} &= \mathtt{STOP} :: \mathtt{ASSERT}_{Kitting} \\
\mathtt{P1} &= \mathtt{SIT}_{Kitting}\langle k\rangle\ :;\ (\\
&\quad\ \tilde{}\,((\mathtt{REASSERT}_{AcquireTray}\langle x1\rangle : \mathtt{REASSERT}_{FillTray,x1} \\
&\qquad : \mathtt{REASSERT}_{FinishTray,x1}) : \mathtt{SUCCEED}_{Kitting,k}) \\
&\qquad\ : \mathtt{FAIL}_{Kitting,k}\ ) \\
\mathtt{P2} &= \mathtt{SIT}_{AcquireTray}\langle k\rangle\ :;\ (\\
&\quad\ \tilde{}\,(\mathtt{REASSERT}_{FindPart}\langle x1\rangle : \mathtt{Move}_{x1,trayarea} \\
&\quad\ : (\mathtt{UPDATE}_{AcquireTray,k,x1} ; \mathtt{SUCCEED}_{AcquireTray,k})) \\
&\qquad : \mathtt{FAIL}_{AcquireTray,k}\ ) \\
\mathtt{P3} &= \mathtt{SIT}_{FinishTray}\langle k, x1\rangle\ :: \\
&\quad\ (\mathtt{STUB}_{FinishTray} : \mathtt{SUSPEND}_{2000} ; \mathtt{FAIL}_{FinishTray,k})
\end{aligned}$$

Forced Assumption Relaxation. Any of the assumption monitors or stub triggers in the initial reactor segments can potentially signal the planner and divert its attention from the a-priori established ordering of assumption relaxation. Failure of a situation to successfully complete its reactions is also cause for a perception. These three sources of perceptual input have different effects on the planner.

On receiving a stub trigger perception, the planner redirects its focus of attention to that portion of the plan containing the stub trigger. An assumption failure perception has a larger effect than simply refocusing attention. This perception causes the planner to negate the assumption in its PSC and begin to rebuild the effected portions of its plan. If a situation has failed that relied on that assumption, that particular situation is given highest priority for adaptation. This may cause the planner to refocus on parts of the plan it had previously considered finished.

Apart from its cause, a forced relaxation or refinement results in the same reactor segment and adaptation sequence as a normal relaxation or refinement. Once the planner has achieved a complete ω-ideal reactor, it is ready again to select the next assumption for normal relaxation.

Experimental Results

The kitting example described earlier was implemented and several (15) experimental runs conducted on a PUMA-560 based kitting workcell. Trace statistics were gathered on each trial run. In each run, the planner utilized all nine assumptions mentioned in section . However, only five of the assumptions were actually relaxed in these trials. The purpose of these experimental runs was to begin to explore the behavior of a planner-reactor system in practice. Results from the runs are presented in the following subsections.

Initial Reactor Construction. Figure 2 contains trace statistics for the startup phase on a typical experimental run of the kitting workcell. Graph (a) shows the number of assumptions in use at any time in the workcell by the planner and by the reactor. Graph (b) shows the number of ***adaptations issued*** (by the planner) and the number of discrete changes to the reactor structure (***adaptations applied***). Initially there is an empty reactor in operation. After 26 seconds (in this example) the planner is in a position to start to update the reactor.

The planner begins by sending adaptations to the reactor (***adaptations issued***). Each adaptation involves one or more changes to the reactor structure. The constraints involved in safe adaptation can cause a time lag in implementing changes. Active situations cannot be interrupted — that would leave the reactor in an undefined state — instead the adaptation waits for the situation to end before applying the changes; the theory behind this is presented in [Lyons & Hendriks 1993]. Thus the ***adaptations applied*** trace always lags the ***adaptations issued*** trace (fig. 2(b)). The lag time varies depending on the activity of the reactor.

The first reactor is in place roughly 28s after startup. Even though this reactor is not complete, the kitting robot can now start kitting, carrying out the initial actions in the "plan"; its kitting actions are overlapped in time with the further refinement of its kitting 'program'. This reactor is elaborated over the following 20s until by 40s after startup the first complete reactor is in place. This reactor employs all nine assumptions.

Assumption Relaxation. Now the planner begins to relax assumptions to incrementally improve the reactor. It takes roughly 5s to relax the first assumption (the length of the 'plateau' in the planner trace in figure 2(a)). Typically one or more adaptations are necessary to relax any one assumption. Each adaptation will give rise to one or more structural changes in the reactor, again subject to the delays of the safe adaptation constraint. In the case of this first assumption relaxation, it takes the reactor roughly 60s before it has been able to implement the last structural change to relax the assumption (the 'plateau' on the reactor trace in fig.2(a)). It takes the planner roughly 30s to relax the second assumption; but the change is implemented in the reactor almost immediately after the first relaxation.

In these experiments, we restricted the planner to only relax two assumptions (AM, ADD). Having done this, the planner will only relax other assumptions when the environment forces their failure. For convenience of presentation we have separated the two kinds of relaxation here.

Environment Forced Relaxation. Figure 3 shows trace statistics for the ongoing operational phase of the kitting workcell. Graph (a) displays the assumptions in use by the planner and reactor, and graph(b) displays the trace of adaptations issued and applied. Additionally, fig.3(b) also shows the trace of assumption failure perceptions. These are the perceptions that force the planner to relax an assumption. The two forced assumption relaxations in fig.3 are the no-motion (AM) and downstream disturbance (ADD) assumptions. The failure perception for AM is generated when the robot fails to acquire a part it had identified in an initial visual image of the workspace. The failure perception for ADD is manually invoked, and would normally be an input signal from downstream automation.

At 152s the AM assumption failure is signaled (Fig.3(b)). The planner responds very quickly and begins to issue adaptations by 155s, continuing for about 10s. There is little delay in the safe adaptation procedure and the structural changes to the reactor lag the adaptations issued by only about 1s. This is also evident from the assumptions trace in (b). One of the changes in this adaptation is to remove the assumption failure monitor for this assumption; thus, partway into the adaptation (161s) the signals from the assumption monitor cease. The total forced relaxation response time — the time from failure to final change in the reactor — is roughly 11s. The ADD failure occurs at 215s and proceeds in a similar fashion.

Reinstating Assumptions. In our theoretical analysis [Lyons & Hendriks 1992b, Lyons & Hendriks 1993] we did *not* address the problem of reasserting previously failed assumptions. Nonetheless, this is convenient in practice to model operating regimes, and therefore is

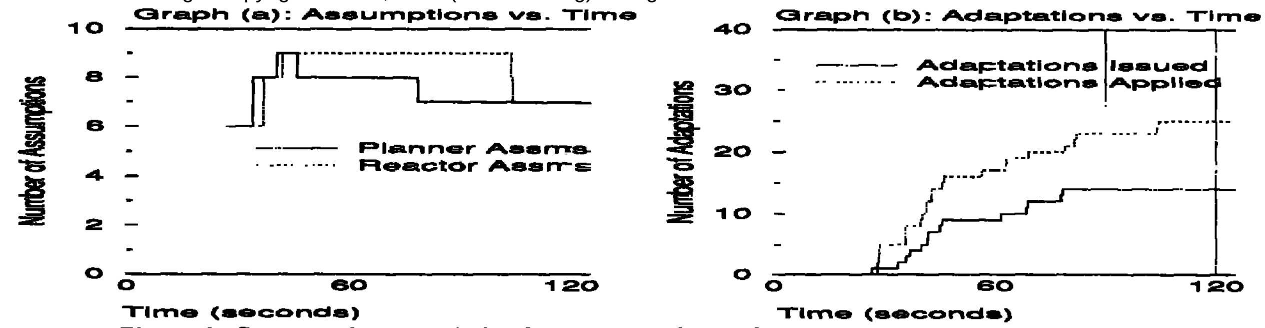

Figure 2: Startup phase statistics for one run of the Planner-Reactor based Kitting Robot.

one of our objectives. In the experimental run, at time 357s the ADD assumption was reinstated (fig.3(a,b)). Upon relaxation of such reinstatable assumptions, the planner adds a reactor monitor that will signal when the assumption holds again. As the trace statistics show, our implementation does indeed handle the reintroduction of assumptions. However, further theoretical work is necessary to extend our convergence results to include this case.

Discussion

ERE. The incremental strategies used in the ERE system [Drummond & Bresina 1990] form a good point of comparision with our assumption-based incremental adaptation algorithm, even though the focus of their work is on once-off activities while ours on repetitive tasks.

In ERE, the *reductor* constructs a behavioral constraint strategy, a partial order on the behavior constraints (temporal goals), based on a causal theory describing the domain. The *projector* uses these to perform a forward beam search from the initial situation to a situation where the first behavioral constraint (BC) is satisfied. This path is compiled into Situation Control Rules (SCRs), IF-THEN rules of the following form: IF situation-i AND behavioral constraint-j THEN do action-i. These SCR's are immediately sent to the *reactor*, where they are the primary action selection mechanism. While the reactor selects and executes actions, the projector continues from the final situation in the path of the first BC to find a path that satisfies the next one (their *cut-and-commit* strategy). If found, new SCRs are generated and sent to the reactor. Once a path is found for all applicable behavioral constraints, the projector starts exploring side paths to increase the robustness of the SCR controlled action selection in the reactor.

The behavioral constraint strategy and our abstract plan are comparable. We chose however to retain the plan structure in the reactor, have all conflict resolution explicit, and have developed explicit 'surgery' procedures to retract or modify reactor segments in a safe manner. In our system, sensing requires conscious actions and must be planned for. The reactor can start new actions asynchronously, and multiple actions may be in progress at one time. In contrast, ERE operates with a select-execute-sense cycle, where an hidden sensory subsystem keeps track of all information, even if it were irrelevant for the current action selection.

The incremental strategies differ mainly in that we use the reactor to guide the planner to the relevant parts to be elaborated on. Through our implantation of **STUB** 's in the reactor, the planner is signalled that the reactor is in need of a segment that has not yet been constructed. Together with the assumption monitors, the planner is kept up to date to the most important parts of the reactor to work on (in response to the particular manifestation of the environment), rather than rely on a predetermined ordering. Through this information flow back to the planner, it can focus on the direct needs of the reactor, generating timely and responsive updates.

Learning. A number of systems in the learning literature have an architecture broadly similar to our planner-reactor system: MD [Chien et al 1991], ERE [Kedar et al 1991], Dyna [Sutton 1990], and Sepia [Segre & Turney 1992] amongst others. The viewpoint from which we designed the planner-reactor system is that there is no learning going on: The planner, which knows implicitly all reactors up to and including the ideal reactor, is simply choosing to express different reactors depending on its perception of the environment. If, on the other hand, the planner's world model was refined over time (as, e.g., in [Kedar et al 1991]), then, we would have argued, there is learning going on.

In retrospect, however, it is clear that the system does contain some aspects of a learning system: the reactor is being incrementally improved based on measurements of the environment. From this viewpoint, the planner-reactor system is similar to an event-based learning system that learns from failure. SOAR [Laird 1990] and MD are examples of other such systems. Failure in our case, though, refers to an assumption failure, not a planning impasse as in SOAR and not (necessarily) a plan failure, as in MD. On this latter point, assumption failures *may* cause action failures, but not necessarily. They may also cause repeated action failures, unlike most learning systems, because of the asynchronous link between planning and reaction. For this reason, as in MD, we assume that the cost of failures in our application domain is much outweighed by the benefits of

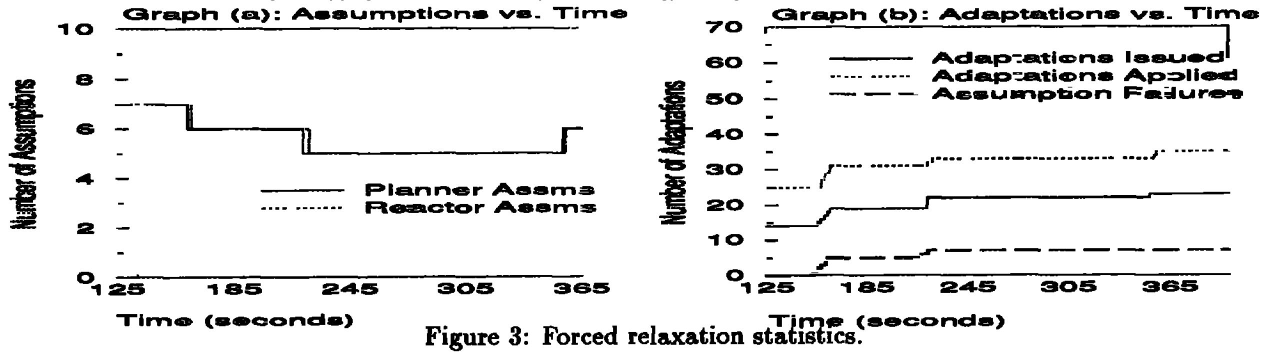

Figure 3: Forced relaxation statistics.

reactivity.

As shown in figure 3, the planner asynchronously begins to refine the reactor when such an assumption failure perception is received and, a short time later, begins to adapt the reactor to its improved version. The planner-reactor differes most from learning systems in its response to an assumption failure. In the terminology expounded by Gratch & DeJong [Gratch & DeJong 1992], the *operationality criterion* that we espouse is based on a predefined linkage between actions and assumptions. When a failure perception is received, the planner can use this predefined linkage to immediately determine which parts of the plan need to be transformed. In an EBL planner such as MD, this is the point at which failure analysis would step in.

Conclusions

In summary, we have reviewed our motivation and approach to casting planning as adaptation of a reactive system and have presented the first performance results of the approach, a first for integrated systems of this kind.

These results show the feasibility of our approach. The incremental reactor construction and assumption relaxation strategy minimizes planning delays. The reactor can start kitting while the planner still is refining later parts of the task. The asynchronous communication between planner and reactor allows the latter to be real-time and not bogged down by the planner. The safe adaptation algorithm guarantees at all times a correct operation of the reactor, while avoiding interruption of non-affected reactor segments.

Further work. We need to extend our theoretical results to include assumption reinstating to model operating regimes. In conjunction, quantitative comparisons with other approaches to build integrated planning-reacting systems are planned.

References

[Bresina & Drummond 1990] 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.

[Chien et al 1991] S. Chien, M. Gervasion, and G. DeJong. On becoming decreasingly reactive: Learning to deliberate minimally. In *9th Machine Learning Workshop*, pp. 288–292, 1991.

[Connell 1992] J. Connell. Sss: A hybred architecture applied to robot navigation. In *IEEE Int. Conf. Robotics & Automation*, pp. 2719–2724, Nice, France, May 1992.

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

[Gratch & DeJong 1992] J. Gratch and G. DeJong. A framework of simplifications in learning to plan. In J. Hendler, editor, *First Int. Conf. on AI Planning Systems*, pp. 78–87. Morgan-Kaufman, 1992.

[Kedar et al 1991] S. Kedar, J. Bresina, and C. Dent. The blind leading the blind: Mutual refinement of approximate theories. In *9th Machine Learning Workshop*, pp. 308–312, 1991.

[Laird 1990] J. Laird. Integrating planning and execution in soar. In J. Hendler, editor, *AAAI Spring Symposium on Planning in uncertain and changing environments*, Stanford CA, Mar. 27–29 1990.

[Lyons & Hendriks 1992a] D. Lyons and A. Hendriks. A practical approach to integrating reaction and deliberation. In J. Hendler, editor, *First Int. Conf. on AI Planning Systems*, pp. 153–162. Morgan-Kaufman, 1992.

[Lyons & Hendriks 1992b] D.M. Lyons and A.J. Hendriks. Planning for reactive robot behavior. In *IEEE Int. Conf. Rob. & Aut.*, Apr. 7–12th 1992.

[Lyons & Hendriks 1993] D.M. Lyons and A.J. Hendriks. Safely adapting a hierarchical reactive system. In *SPIE Int. Rob. & Comp. Vis. XII*, Sept. 1993.

[Lyons 1993] D.M. Lyons. Representing and analysing action plans as networks of concurrent processes. *IEEE Trans. Rob. & Aut.*, 9(3), June 1993.

[McDermott 1991] D. McDermott. Robot planning. Tech. Rep. YALEU/CSD/RR#861, Yale, Aug. 1991.

[Schoppers 1989] M. Schoppers. *Representation and Automatic Synthesis of Reaction Plans.* Tech. Rep. UIUCDCS-R-89-1546, Dept of Comp. Sc., Univ. Illinois, 1989.

[Segre & Turney 1992] A. Segre and J. Turney. Sepia: A resource-bounded adaptive agent. In J. Hendler, editor, *First Int. Conf. on AI Planning Systems*, pp. 303–304. Morgan-Kaufman, 1992.

[Sutton 1990] R. Sutton. First results with Dyna. In J. Hendler, editor, *AAAI Spring Symposium on Planning in uncertain and changing environments*, Stanford CA, Mar. 27–29 1990.