Integrating Planning, Execution and Monitoring*

José A. Ambros-Ingerson
Dept. of Info. and Computer Science
University of California
Irvine, CA 92717
jambros@ics.uci.edu

Sam Steel
Dept. of Computer Science
University of Essex
Colchester, CO4 3SQ
United Kingdom

Abstract

IPEM, for Integrated Planning, Execution and Monitoring, provides a simple, clear and well defined framework to integrate these processes. Representation integration is achieved by naturally incorporating execution and monitoring information into [Chapman, 1987] TWEAK's partial plan representation. Control integration is obtained by using a production system architecture where IF-THEN rules, referred to as flaws and fixes, specify partial plan transformations. Conflict resolution is done using a scheduler that embodies the current problem solving strategy.

Since execution and plan elaboration operations have been designed to be independently applicable, and execution of an action is a scheduling decision like any other, the framework effectively supports interleaving of planning and execution (IPE). This renders a local ability to replan after both unexpected events and execution failure.

The framework has served as the basis for an implemented hierarchical, nonlinear planning and execution system that has been tested on numerous examples, on various domains, and has shown to be reliable and robust.

1 Introduction

As early as 1974, [Sacerdoti, 1974] writes "[F]or a system that deals with complex problems in a real world, as opposed to a simulated one, it is undesirable to solve an entire problem with an epistemologically adequate plan. There are too many reasonably likely outcomes for each real-world operation." (133) Further on he suggests that this can be achieved in a hierarchy of abstraction spaces where "[T]he process of alternatively adding detailed steps to the plan and then actually executing some steps can continue until the goal is achieved." (134)

This problem solving strategy needs a framework that allows interleaving planning and execution, and furthermore, a control policy to indicate when to plan and when to execute.

*The research reported herein has been partly supported by an Overseas Research Student Award by the CVCP of the UK (ORS/85281); the Teamwork Project funded by the SERC of the UK (GR/C/44938); and Hewlett Packard México through a fellowship administered by UC-Mexus. This research has *not* been supported by a military agency. Any use of the results presented here for military purposes is contrary to the intentions of this research.

IPEM is an attempt to provide such framework, not only to support Interleaving of Planning and Execution (IPE) but also to support replanning in dynamic environments where unexpected events can occur, and where actions can fail to bring about their intended effects.

We should note that IPE is present in replanning, planning in dynamic environments, plan repair, etc. If a system is to execute its plans, IPE will be the norm and not the exception.

The present document presents an overview of the IPEM framework and system. For a detailed description please see [Ambros-Ingerson, 1987].

2 Related Work

IPEM relates to other work in the field along three important dimensions:

1. *The representation used for actions and plans.* This relates it to planning systems like STRIPS [Fikes and Nilsson, 1971], NOAH [Sacerdoti, 1974], NONLIN [Tate, 1977], and more recently, TWEAK [Chapman, 1987].
2. *The control mechanism used in the elaboration of the plan.* The great majority of systems use fixed control strategies. Alternatives explored have been MOLGEN [Stefik, 1981], Bartle's Cross-Level Planning [Bartle, 1986] and, in Blackboard Architectures, the use of a task scheduler as in HEARSAY-II [Lesser and Erman, 1977], which is the approach taken by IPEM and in Tate's O-Plan system [Currie and Tate, 1985].
3. *The execution monitoring and replanning capabilities.* Very few planning systems execute their plans (either controlling some robot or in a simulated environment) and consequently aren't faced with this problem. Of those that do the most relevant are PLANEX [Fikes, 1971] (the execution module for STRIPS), NASL [McDermott, 1978], ELMER– a taxi driver in a simulated city– [McCalla and Reid, 1982], Phil Hayes's work on replanning using dependency records [Hayes, 1975], and [Wilkins, 1985] addressing the issue of recovering from execution errors in SIPE. More recently attention has been devoted to reactive planning; e.g., the work of [Georgeff and Lansky, 1987] on procedural logic and [Schoppers, 1987] on universal plans.

3 IPEM: Framework and Implementation

The IPEM system was designed with the goal of supporting interleaving planning and execution. An integrative approach requires that both execution and planning decisions be based upon and recorded on a common representation. We use a partial plan representation similar to the one used in other systems (e.g., TWEAK), extended to include the current world description, the actions in the process of being executed, etc. We also maintain a decision list that records the history of the problem solving process (e.g., for backtracking).

A problem solving strategy that interleaves planning and execution should not be constrained by dependencies between planning and execution operations. Thus, all our transformations – the flaws and fixes that are used to elaborate and execute a plan – were designed to preserve the well-formedness and semantics of partial plans and can be applied independently of each other.

Our bias has been to design plan transformations that are clean, simple and composable, instead of powerful ones, that are often complex and ad-hoc. Complex powerful transformations are obtained by the application of a sequence of simple ones.

We use a production system architecture since it provides the flexibility in control that we need [Lesser and Erman, 1977]. IF-THEN rules map to flaws and fixes in our framework. A flaw is a property or condition in a partial plan that corresponds to the IF part of an IF-THEN rule; each fix – there is usually more than one – corresponds to the THEN part, and specifies a plan transformation to get rid of (i.e., fix) the flaw. Conflict resolution is done using a scheduler that embodies the current problem solving strategy along with weak and domain specific heuristics. Alternative options at a choice point are retained so that full backtracking is supported whenever possible.

IPEM has been implemented in C-Prolog at Essex (Sun3/50 and GEC-63) as the core of a multi-actor planning and execution system [Doran, 1987]. It allows the user to input unexpected events at any stage of execution and plan development. The examples presented here, and others that involve interactions in a multi-actor setting [Doran, 1986], run satisfactorily. The system is currently being used at Essex for research in plan delegation and organization emergence [Doran, 1988].

3.1 Assumptions

Our action representation is – by historical accident, since it was developed independently – almost identical to Chapman's TWEAK, so all his assumptions are our assumptions (e.g., STRIPS assumption).

We further assume a continuously updated Current World Description (CWD), in the form of a set of ground (i.e., variable free) propositions. We do not assume the description is complete but we do assume it contains no errors of commission. We don't make the closed world assumption. Note however, that:

- we do not assume a static world – the CWD can change while the plan is being elaborated, possibly making the current partial plan inapplicable;
- actions can fail to achieve its intended effects; partial success is exploited;
- actions are not assumed to achieve their effects immediately – in fact, different effects of the same action can have different delays without preventing the execution of those parts of the plan that can be safely executed, and;
- we don't assume the CWD holds all the information needed to elaborate a complete, detailed plan at planning onset – so the problem may be unsolvable without interleaving planning and execution.

We say that an incomplete (partial) plan P *necessarily* satisfies property S if S holds in every possible completion (elaboration) of P. It *possibly* satisfies S if there is at least one completion that satisfies S. See [Chapman, 1987] for more details.

3.2 Plan Elaboration

The plan transformations used to elaborate the plan are very similar to those used in other systems (e.g., NONLIN, TWEAK). We will only give a brief description here.

A (well-formed) partial plan[1] consists of:

1. a partially ordered set of actions including two special ones, BEGIN (the minimum) and END (the maximum) – the postconditions (effects) of BEGIN are the propositions in the CWD and the goals to be achieved are the preconditions of END.
2. a set of (protection) ranges – each range connects a postcondition (supplier) with a precondition (consumer) indicating the (sub)goal dependency and requiring both propositions to necessarily codesignate (necessary codesignation is equivalent to unification). Note that the supplier has to be necessarily before the consumer.

The initial partial plan has of two actions, BEGIN and END, to which the following transformations are applied.

3.2.1 Unsupported Precondition; Reduce

An action A in the plan with a precondition with no range (i.e., it has no assigned producer) has an *unsupported precondition* flaw. The postcondition to be used as producer for the new range can be assigned in any of three ways:

1. by simple establishment on an action already in the plan which is necessarily before A (*reduction prior*), or
2. by simple establishment on an action B already in the plan which is possibly before A (*reduction parallel*). B is now constrained to be necessarily before A; or
3. on a new step (action) now added to the plan (*reduction new*).

3.2.2 Unresolved Conflict; Linearize

A plan with a range R that protects proposition p and an action A, possibly after the producer and possibly before the consumer of R that asserts the negation of q, where p and q necessarily codesignate, has an *unresolved conflict* flaw (clobbering). This is fixed by promotion (A is constrained to be necessarily after the consumer of R) or demotion (A is constrained to be necessarily before the producer of R). In both cases the plan is partially *linearized*.

3.2.3 Unexpanded Action; Expand

A plan with an action which is expandable (i.e., not primitive) has an *unexpanded action* flaw. Actions and their expansions up and down the hierarchy are linked

[1]See [Ambros-Ingerson, 1985] for a detailed definition of well-formed partial plan.

by their pattern – an *action based description* (e.g., 'dance *style*') – and codesignation of patterns constrain variable bindings in the same way as with ranges. Expansion consists of replacing such action (together with the ranges attached to it) with an appropriate expansion instance – a partial plan in itself (e.g., a sequence of foot moves that realize the dance).

This contrasts with the *state based description* expressed through pre and post-conditions that reduction uses (e.g., an action that takes '*foot@loc1*' to '*foot@loc2*').

3.2.4 Completeness and Correctness

A comparison between IPEM's and TWEAK's plan transformations shows that IPEM has no fixes equivalent to separation and white night while TWEAK has no action expansions. Can we claim IPEM complete and correct?

Correctness follows from the fact that both IPEM and TWEAK detect the same set of flaws. Provided expansion schemas are correct, their inclusion in the plan can't generate incorrect plans (although they can certainly introduce new flaws). In fact, action expansions can be defined in terms of a sequence of reductions and linearizations.

Although our clobbering (unresolved conflict) definition is narrower than TWEAK's (it requires necessary instead of possible codesignation), we can show that they only yield different results on plans that are complete except for unbound variables in the post-conditions of some action(s). There is more than one way to define the semantics of executing such an action. We return to this issue in Section 5.

Completeness follows from noticing that a white knight fix is equivalent to replacing a range with a far producer for one with a closer one. This transformation can be avoided by selecting the final producer correctly in the first place. The same argument holds for separation; it can be avoided by selecting bindings right in the first place. Note that the completeness claim has to be dropped if unexpected events or execution failure is allowed, since the notion is ill defined in this case.

What *can* be affected is the efficiency of plan generation. If used with the same search regime, IPEM will probably backtrack more often because it posts more stringent constraints than it needs to. Our selection of few fixes however, matches our bias for simplicity referred to previously.

3.3 Plan Monitoring and Execution

We extend the action representation to accommodate the necessities of execution and monitoring. We associate a procedure and a time-out with every primitive action as explained below.

3.3.1 Unsupported Range; Excise Range

A plan with a range R produced by BEGIN, protecting a proposition no longer in the CWD, has an *unsupported range* flaw. Note that ranges produced by BEGIN are precisely those propositions in the CWD that are currently relied upon (assumed) by the partial plan.

Excising R fixes this flaw but automatically creates an unsupported precondition on R's consumer. On the other hand, since the codesignation constraint is also removed (it is part of the range) new bindings might be permissible.

The effect of the REINSTANTIATE operator in SIPE [Wilkins, 1985] is analogous to the application of an excise range followed by a reduction prior on BEGIN. We return to this relationship in Section 4.

3.3.2 Unexecuted Action; Execute

A plan with an action ready for execution has an *unexecuted action* flaw. We consider an action A ready for execution if

- A is primitive and not END;
- all its preconditions have ranges produced by BEGIN, none of which is unsupported;
- it is immediately after an executed action (BEGIN is considered executed);
- it is not involved in an unresolved conflict flaw; and
- there is no "live" action B (i.e., not timed-out) before A that expects a post-condition that can clobber any of A's post-conditions.

Executing the action consists of :

- adding order constraints so that all parallel actions are made necessarily after A;
- calling the associated procedure (after substitution of the appropriate bindings) which in turn should instruct some effector to carry out a movement, a measurement, etc.;
- if active monitoring by a lower level system is desired, posting the action's postconditions as expected; and
- recording that the action has been executed.

Note that the action is kept in the plan (see time-out below).

Although this execution model does not allow simultaneous execution initiation, it does allow the execution initiation of actions before their predecessors have finished (timed-out). Thus, if the planner's cycle is fast with respect to execution completion times, actions executing in parallel can be present.

3.3.3 Timed Out Action; Excise Action

An executed action *times out* when no further effects are expected to come about as consequence of its execution. Note that time-out is not defined in terms of success or failure; every action must time-out, whether it achieved its intended effects or not.

Fixing this flaw entails removing the action – together with all the ranges for which some postcondition is a producer or some precondition a consumer – from the plan.

We previously pointed out that actions are kept in the plan when executed. It's important to note that this is consistent with the semantics of an action in a partial plan. In fact, since incorporating the expected effects into the postconditions of BEGIN would violate the semantics of the CWD, deleting the action from the partial plan would necessitate the creation of new structures to record the expected effects and their interaction with other parts of the plan. However, this is what planning is all about and is done for every action in the plan. So why duplicate this effort in another structure? An unexecuted action and an executed one that hasn't timed-out are both predicting a future state of affairs over which the same kind of planning reasoning applies.

3.3.4 Unextended Range; Extend Range

If, in a plan, we can remove a range R and replace it with a range R' where

- R and R' have the same consumer;
- the producer of R' is before the producer of R; and
- the new range R' does not create an unresolved conflict flaw (clobber);

we have an *unextended range* flaw. The fix is to replace R with R', which can be seen as extending range R on the producer side to the location of the producer of R'; hence its name: *range extension.*

We distinguish the special case where the producer of the new range is BEGIN and the consumer is a post-condition of an executed action A. In this case the appearance of the proposition at BEGIN that makes the extension possible can be correlated with a (partially, at least) successful execution of A.

The general case where no execution has taken place can be considered to be a serendipitous occurrence, especially if the extension is to some postcondition of BEGIN. Range extensions have to be done with care in the general case (only) since there is a tradeoff between having long ranges, that are more likely to generate interactions, and short ones, that take no advantage of serendipitous occurrences to remove redundant actions.

3.3.5 Redundant Action; Excise Action

A plan with an action A not producing for any range (i.e., no range has A as producer) has a *redundant action* flaw. Excising the action – its fix – can result in further action redundancies since ranges are removed along with the action.

3.4 Control: The Scheduler

Conflict resolution for the application of a given fix at a given moment is done through a scheduler similar to the one used in HEARSAY-II [Lesser and Erman, 1977]. It maintains an agenda (priority queue) of tasks. Each task consists of a flaw, along with its possible fixes. Tasks and fixes for a flaw are ordered using weak and user supplied domain heuristics.

Although the search space is not a strict AND-OR graph, we use some of the heuristics that work well there. Since all flaws have to be fixed, we select the one that is "harder" to fix, and select the fix that introduces less constraints into the partial plan (e.g., in the case of reductions, the preference order we use is prior, parallel and new).

In our runs we have set to fix flaws in the following order: unsupported range, unextended range, timed-out action, unresolved conflict, unsupported precondition, unexpanded action, unexecuted action. Within a flaw class, the flaw which appears harder to fix (e.g., has only one fix) is preferred. If some flaw has no fixes, then backtracking is attempted when possible.

3.4.1 Backtracking

The system uses full chronological backtracking up to decision points that involve non-backtrackable fixes (e.g., excision of an unsupported range caused by an unexpected event, action execution, etc.). Beyond this point two general approaches can be taken, where the choice is domain and case dependent. Note however, that all replanning options will be attempted before backtracking is chosen.

The first is to ignore such decision points (they have only one fix anyway) and carry on up the tree to other choice points with open alternatives. Note however that completeness is compromised; e.g., it is possible that the very same option currently being backtracked over is now viable thanks to a just occurred unexpected event.

Alternatively, we can scrap the old plan and start afresh on a new plan with the same goals. Simplicity makes this approach more appealing.

4 Unexpected Events and Replanning

This section will illustrate IPEM's replanning adaptability to both a dynamic world and to execution failure. This example is very similar to the one presented by Wilkins [Wilkins, 1985] and fits the scenario proposed by Schoppers [Schoppers, 1987], where a mischievous baby makes the state of the world all but static.

The goal is to achieve ((a on c) $\wedge$ (u.x on r.y)) from the blocks configuration presented in Figure 1:i. The initial plan (move a from b to c) in parallel to (move u.2 from t.5 to r.1) is at the top left labeled 'i'.

Before execution of this plan can commence, our baby (moves d from t.3 to r.1). This causes the addition of ((clear t.3) $\wedge$ (d on r.1) $\wedge$ -(d on t.3) $\wedge$ -(clear r.1)) to the CWD creating an unsupported range flaw on (clear r.1). The range is excised, creating an unsupported precondition on (clear z), which is fixed by reduction prior on (clear r.2) (see ii). The overall effect is very similar to the REINSTANTIATE operator described by Wilkins [Wilkins, 1985].

At this point our baby interferes again, (moving a from b to u.2). This interferes with both actions in the plan. Two unsupported ranges result; for (clear u.2) and for (a on b). The ranges are excised, generating the corresponding unsupported preconditions for (clear u.2) and (a on y). The first one is fixed by reduction parallel to the effect (clear y) of the action moving block a (now necessarily before as a consequence of the constraint that for a given range, the producer is necessarily before the consumer). The second one is fixed with a reduction prior to (a on u.2) at BEGIN. The resulting plan is shown at 'iii': (move a from u.2 to c) followed by (move u.2 from t.5 to r.2).

There is an unexecuted action flaw on the first action, now fixed by execution. The effector starts by picking block a up. (clear u.2) is added to the CWD, generating an unextended range flaw, fixed by extension. Now the effector screws up and bumps into d and r.1 producing the state shown in 'iv'. Now the action times-out (say the manipulator sends a signal indicating it bumped into something and that it is no longer attempting to carry out the procedure) and must be excised. This generates an unsupported precondition at (a on c) of END. Note that execution of (move a from u.2 to c) was *partly* successful and that IPEM took notice; execution of (move u.2 from t.5 to r.2) could have proceeded if another manipulator were available even though the other action hadn't timed-out yet.

A number of other propositions are added to the CWD as a consequence of the bumping (e.g., (clear r.1), (d on c), -(a on u.2), etc.). The unsupported precondition for (a on c) is now fixed with two reduction new fixes and a number of reduction priors. The resulting plan is shown in 'iv'.

It is important to point out the adaptability displayed by IPEM under unexpected events and execution failure. This it shares with reactive planners (e.g., [Schoppers, 1987],

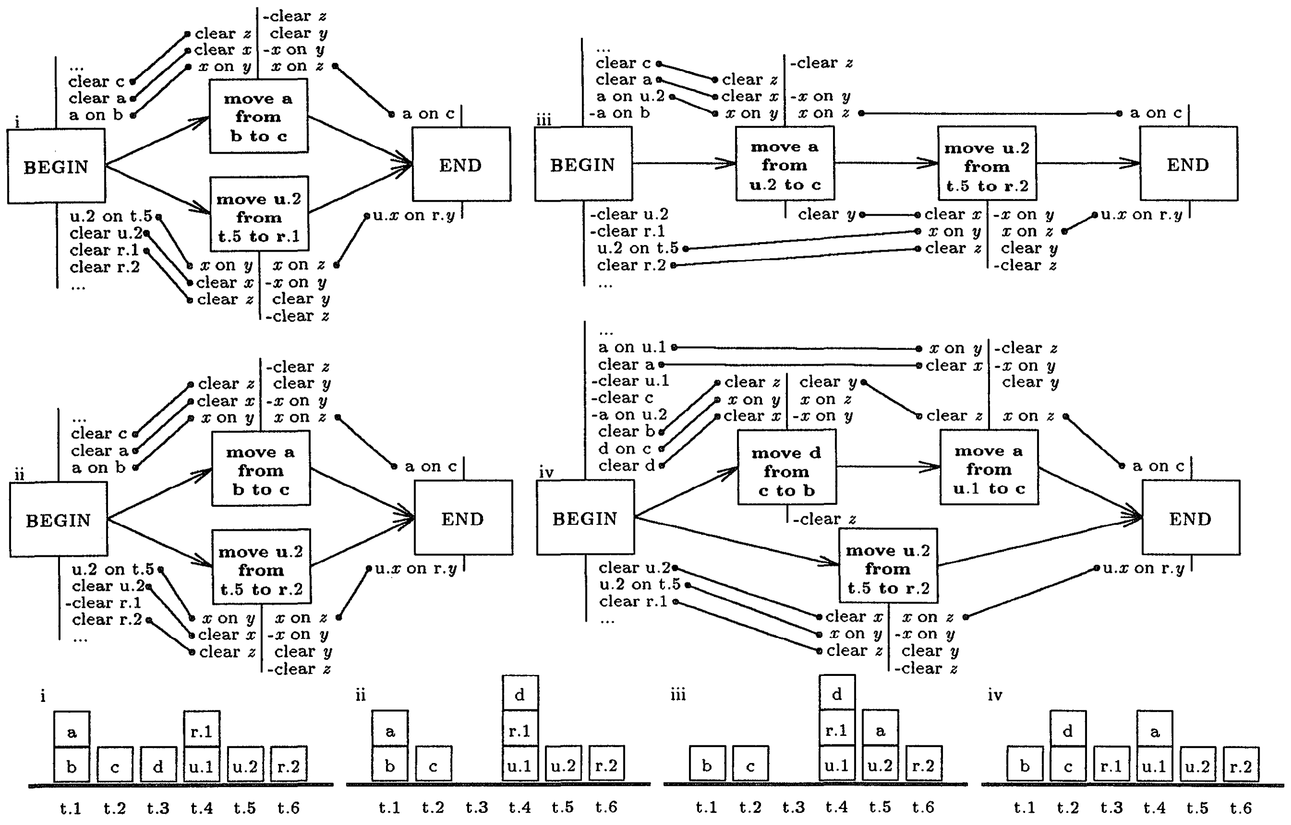

Figure 1: Replanning after unexpected events and execution failure.

[Firby, 1987]). On the other hand it retains the flexibility and power of a hierarchical nonlinear planner. It is capable of dealing with action interactions where reactive planners are incapable of doing so.

5 Interleaving Planning and Execution

> We are having dinner with Sandy. When having dinner with somebody we want that somebody to *like* the food and we also *need* to have the food. We don't know what Sandy likes but it does make a difference since the choice determines the restaurant, the way to dress, whether to make a reservation, etc. Asking is a way of finding what a person likes.

This example illustrates a class of problems that can hardly be solved without interleaving planning and execution. A conditional plan is a poor option since it must plan for a potentially large number of alternatives, most of which won't be used. On the other hand, replanning after execution failure can be harmful to our relationship with Sandy.

To solve this problem using IPEM we need to make an extension to the action representation to cope with information acquiring actions (IAA's). TWEAK's semantics define an otherwise complete plan with an unbound variable on the postconditions of an action to be completed to *any* constant. This is unsatisfactory for an action which yields a particular – yet unknown – value when executed.

We introduce a new set of variables, *Ivar* to handle such actions. The variable in the postcondition intended to provide information in an IAA is defined as an Ivar. We allow Ivar's to codesignate with variables, but not with constants. The scheduler is slightly modified to put on hold those flaws whose fixes would bind an Ivar to a constant. This results in plan elaboration up to the point where all flaws are on hold with the exception of unexecuted actions, then chosen for execution by the scheduler.

For example, consider Figure 2. The top presents a plan to solve our dining example problem, where *thing* is an Ivar at ASK, our IAA action. At this point every expansion for GET *meal* in the action schema repertoire known to the system would bind *meal* to some constant. Since *meal* codesignates with *thing* its expansion is placed on hold. Note that this is the maximally elaborated plan that doesn't commit the binding of *meal*.

Unexecuted action at ASK is the only remaining flaw not on hold. ASK is executed and made necessarily before GET *meal* (see bottom of Figure 2). "Sandy likes fondue" is eventually added to the CWD creating an unextended range from DINE to BEGIN. The range is extended (the solid range added, the dotted one removed) so that *meal* now codesignates with a constant (fondue) and not with *thing*. Planning can proceed now that it has been established that the correct expansion for GET *meal* is GET fondue.

Note that an action has been executed before the plan was fully elaborated and the outcome of its execution was used to decide the expansion to use (i.e., for a planning

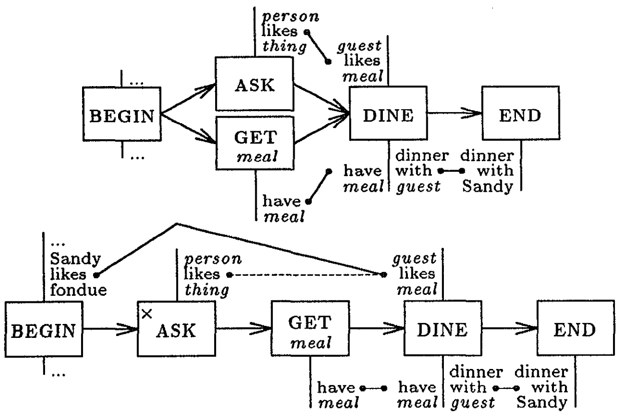

Figure 2: Interleaving Planning and Execution

decision).

Conclusions

IPEM successfully integrates the processes of planning, execution and monitoring. Control integration was obtained by using a production system architecture where the IF-THEN rules operate as transformations between partial plans. In IPEM's context IF-THEN rules are referred to as flaws and fixes. Representation integration was achieved by using a common partial plan representation extended to include monitoring and execution information.

The primary goal of providing a system to support interleaving of planning and execution in a principled and clear way has been attained.

Furthermore, the system exhibits a robust capacity to replan either after execution failure or after the occurrence of unexpected effects. One must caution however, that this capability is seriously limited by its locality.

A planning and execution system embodying the framework has been successfully implemented. It has been tested on numerous examples from various domains, including the ones in this document. It is currently being used at Essex in research concerning Multi-Actor systems [Doran, 1988].

Acknowledgements

Many thanks must go to Jim Doran for support, encouragement and valuable insights and suggestions. We also thank David Aha, Tony Lawson, Chris Trayner, Edward Tsang and Wayne Wobcke for useful comments in the course of this research.

References

[Ambros-Ingerson, 1985] J. A. Ambros-Ingerson. *Planning; a Theory, an Application and a Tool.* Master's thesis, University of Essex, Colchester CO4 3SQ, U. K., 1985.

[Ambros-Ingerson, 1987] J. A. Ambros-Ingerson. IPEM: Integrated planning, execution and monitoring. M. Phil. Dissertation, University of Essex, Colchester CO4 3SQ, U. K., 1987.

[Bartle, 1986] R. A. Bartle. *Three Ways to Cross-Level Plan.* Technical Report CSM-84, University of Essex, University of Essex, Colchester CO4 3SQ, U.K., 1986.

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

[Currie and Tate, 1985] K. Currie and A. Tate. *Control in the Open Planning Architecture.* Technical Report AIAI-TR-12, Artificial Intelligence Applications Institute, Edinburgh, U.K., 1985.

[Doran, 1986] Jim E. Doran. *Distributed Artificial Intelligence and the Modelling of Sociocultural Systems.* Technical Report CSM-87, University of Essex, Colchester CO4 3SQ, U.K., September 1986.

[Doran, 1987] Jim E. Doran. *A Computational Investigation of Three Models of Specialisation, Exchange and Social Complexity.* Technical Report, University of Essex, Colchester CO4 3SQ, U.K., August 1987.

[Doran, 1988] Jim E. Doran. The structure and emergence of hierarchical organisations. In *Alvey Workshop on Multiple Agent Systems*, Philips Research Labs, Redhill U.K., April 1988.

[Fikes, 1971] Richard E. Fikes. *Monitored Execution of Robot Plans Produced by STRIPS.* Technical Note 55, Stanford Research Institute, 1971.

[Fikes and Nilsson, 1971] 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.

[Firby, 1987] R. J. Firby. An investigation into reactive planning in complex domains. In *AAAI'87*, pages 202–206, 1987.

[Georgeff and Lansky, 1987] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *AAAI'87*, pages 677–682, 1987.

[Hayes, 1975] P. J. Hayes. A representation for robot plans. In *IJCAI'75*, pages 181–188, 1975.

[Lesser and Erman, 1977] V. R. Lesser and L. D. Erman. A retrospective view of the Hearsay-II architecture. In *IJCAI'77*, pages 27–35, 1977.

[McCalla and Reid, 1982] G. I. McCalla and L. Reid. Plan creation, plan execution and knowledge acquisition in a dynamic microworld. *International Journal of Man Machine Studies*, 16:189–208, 1982.

[McDermott, 1978] Drew McDermott. Planning and acting. *Cognitive Science*, 2:71–109, 1978.

[Sacerdoti, 1974] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

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

[Stefik, 1981] M. Stefik. Planning and meta-planning (Molgen: part 1 and 2). *Artificial Intelligence*, 16:111–170, 1981.

[Tate, 1977] Austin Tate. Generating project networks. In *IJCAI'77*, pages 888–893, 1977.

[Wilkins, 1985] D. E. Wilkins. Recovering from execution errors in SIPE. *Computational Intelligence*, 1:33–45, 1985.