From: AAAI-82 Proceedings. Copyright ©1982, AAAI (www.aaai.org). All rights reserved.

ERROR DETECTION AND RECOVERY IN A DYNAMIC PLANNING ENVIRONMENT

Blake Ward and Gordon McCalla

Dept. of Computational Science
University of Saskatchewan
Saskatoon, Saskatchewan, S7N 0W0

ABSTRACT

In this paper a set of techniques for error detection and recovery is proposed. These techniques augment a planning system (the ELMER system) which already has many features for preventing execution errors but has few features for handling errors that can't be prevented. The error handling techniques presented in this paper depend for their effectiveness on the close co-operation of the planning, execution and knowledge base components of the system, and especially make use of knowledge learned from the earlier execution of other plans.

1. INTRODUCTION

For the past several years we have been developing a geographic planning system which takes the view that planning, execution, and knowledge acquisition are inseperable components of a planner operating in a dynamic environment. The ELMER system (McCalla et al [1978],[1979],[1982]) uses a unique route-splicing planning methodology to produce plans to take a taxi driver (ELMER) to a destination in a small (simulated) city (Figure 1) which contains dynamic obstacles such as cars, pedestrians, etc. These obstacles are anticipated well enough in the current system that error recovery in seldom needed. However, for some dynamic situations, errors will happen and this paper explores how the ELMER system can be extended to handle error recovery.

Other research besides ours has emphasized the importance of handling execution errors. HACKER (Sussman [1973]) for example produces plans and then debugs them. Srinivas [1977] categorizes errors which occur in the execution of robot plans and suggests several approaches to correcting them. The incremental planning system of Hayes-Roth and Hayes-Roth [1979] suggests that people don't produce perfectly structured plans and then execute them, but take a much more integrated approach where plans are produced and then modified and corrected through simulated execution. Other systems discuss the importance of execution monitoring and error correction, but don't necessarily suggest they be done automatically. One such system is the interactive planning system of Robinson and Wilkins [1980] where monitoring and execution are done by the user and not the planner. Thus the addition of an error recovery capability to the ELMER system is important, and the techniques outlined here may have wider applicability than just ELMER.

2. THE ELMER SYSTEM

The ELMER system has three major components: the **Planner**, the **Map**, and the **Executor**, as shown in Figure 2. The Executor receives **windows** of low-level sensory information from the geographic microworld indicating the presence of both permanent features (stop signs, street signs) and transient features (cars, pedestrians) at varying distances from ELMER. It attempts to correlate this window information with the hierarchical plan it is trying to execute. For example, the core plan in Figure 3, (ie. the numbered plans headed by plan 23) represents a plan to go from the intersection *Reiter @ Winograd* to *Schubert @ Brachman* (streets in a mythical city in which ELMER "exists"). Plan 23 breaks down into sub-plans 24 and 25 (representing traversals of smaller portions of the path) and these in turn break down to sub-sub-plans 120, 121 and 122 and 123, 124 and 125 respectively. The Executor activates the left-most branch of the hierarchy (23, 24, and 120) and looks for window information which allow **transitions** (eg. *Past Reiter @ Winograd*) to be made from any of these active plans. For example, if *At Reiter @ Schubert* is recognized (say by the presence of an appropriate street sign), the new active branch of the hierarchy will become 23, 24, and 123. Certain predictable errors which could arise if stop signs or red lights or the like were missed can be prevented by **secondary** plans such as A,B,...,H which act in parallel to the core plan throughout its execution.

Once the plan has been successfully executed, the plan is added as a **route** to the map. Routes are just plans instantiated by attaching the window seen during execution beneath the primitive plans where they were "seen". Certain information (such as speed limits, directions, distances, etc.) can be abstracted into higher level plans, as well, and information about how this new route connects (**associates** or **inassociates**) to other routes is also added. Intuitively, a route associates into another route if it is "easy" to get from the first route to the second. At the same time the second route is said to inassociate to the first.

Such associations/inassociations form the basis of the Planner's ability to create plans. When presented with the request to "go from A to B", the Planner first of all looks to see if the Map has a route already connecting A to B (ie. ELMER has made this trip before). Note that this route only needs to be part of a previous trip--extra portions before A or after B are ignored. If not, it looks to see if there is a route connecting some route that A associates into to some route inassociated to B. (ie. if it is easy to get to some place close to A from which it is easy to get to some place close to B). If not, associations/inassociations at higher levels of detail are tried, and if this still fails, the Planner attempts to splice together two routes, the first of which *contains* A, the second of which *contains* B and which mutually intersect at some point. Once an appropriate plan has been concocted it is passed down to the Executor.

There are many aspects of the ELMER architecture which have been ignored (eg. secondary plans) since they aren't needed for the discussion to ensue. Further details can be found in McCalla et al [1982].

3. ERROR HANDLING

Error handling in the original ELMER system was downplayed as a problem. Instead, secondary plans tried to prevent errors from ever occurring by explicitly predicting certain dynamic situations (red lights, pedestrians in the way, etc.). There are many errors, though, that can't be explicitly predicted – running into unexpected road blockages, missing some vital road sign, using a route that doesn't quite go as expected. These kinds of errors will occur with increasing frequency as the Map is extended to infer hypothetical routes and as realistic resource constraints are placed on planning and execution.

3.1 Error Detection

The first problem to be tackled when trying to recover from execution error is even recognizing that an error has occurred. This is non-trivial in the ELMER system (as it often is in the real world). The difficulty is in determining when a transition out of a plan has **not** occurred as expected. There are two basic approaches to error detection:

i. the Planner can explicitly add **error transitions** to plans, the ELMER analogue to saying "if you see the drug store on your right you've gone too far"; or

ii. the Executor can **monitor** the execution of the plan watching for any of several conditions that indicate an error may have occurred.

Explicitly added error transitions come from two sources. The primary source comes as a residue of the Planner's route splicing methodology. Basically, if we wanted a plan to go from *Winograd @ Schank* to *Reiter @ Schubert* and the Map currently contained the routes shown in Figures 3 and 4 then the resulting plan would be that shown in Figure 5. Notice that in splicing the two routes together, the Planner has added two new transitions that indicate error conditions. The Planner does have to use some care in adding error transitions since it is possible that two routes that are being spliced overlap and share a common sub-route. Since the overlapping portion is being used as a part of the final plan, it cannot be added as an error transition as well.

Another source of explicit error transitions comes as a result of errors made in the execution of a previous plan. When the old plan is being added to the Map, it is relatively straightforward to abstract a record of the erroneous path previously traversed and tack it on to the old plan as a transition from the sub-plan where the error originated. The planner can then pass any such relevant error transitions down to the Executor so that the error can be instantly recognized in the future.

If the error can't be recognized via explicit error transitions, it must be detected by monitoring plan execution. One approach is to compare the windows being seen during execution to those seen on previous traversals of the same route. To accomplish this, the Executor must have access to windows (or information abstracted from windows) that the Map has kept after previous traversals of the route. As a plan is executed, each window can be matched with the appropriate small set of previously seen windows beneath the current primitive plan (or can be searched for features corresponding to more abstract information). Domain specific heuristics need to be employed in order to match only the relevant features. In the ELMER world, permanent features such as street signs and and buildings are important while more transient features such as other cars and pedestrians are not. If crucial features don't match then an error probably has occurred.

An alternate approach to error detection involves adding a distance attribute to each go-along box in the Map that records the total length of the stretch of road traversed by that plan box. When the distance travelled by ELMER since the transition into that box exceeds the recorded distance an error has definitely occurred.

3.2 Error Recovery

Once the error has been detected, recovery must take place. This largely boils down to trying to figure out where ELMER is now. If all else fails, ELMER can ask the dispatcher, but there are several heuristic approaches to solving this problem. One approach (somewhat like the plan patching of Srinivas [1977]) suggests **retracing** the steps taken when the error occurred in order to get back to the original plan. In the case of explicit error transitions, this may be fairly straighforward since the erroneous path is already part of the plan and need only be reversed in order to get back on track. Of course, one-way streets may foil this attempt, so there are no guarantees even in this simple case. Retracing may also be possible by abstracting primitive level plans from windows seen as the erroneous path was undertaken and then reversing the order of these plans. Not every window will contribute to a low-level plan nor is it always the case that a needed piece of information will be in a window, but the approach will often work. Once the path has been retraced, getting to the original destination is usually straightforward since the original plan can now be re-activated, unless, of course, road closures or the like continue to make the original plan inappropriate.

Under such circumstances or in situations where it is impossible to retrace the wrong steps taken or when such retracing is impracticable (eg. owing to the length of time to achieve it), other methods must be sought. One approach is to **find a nearby location** which is in the Map, try to get there, and then ask the Planner to re-plan a path to the destination. If the current location corresponds to an intersection known to the Map, then the problem is trivial--just re-plan from there.

If not, ELMER is (in a sense) lost in that his position at the primitive levels is unknown to the Map. However, windows attached to the primitive level plans may be recognizable. Since ELMER probably hasn't gone far wrong, it is possible to look at routes associated/inassociated to the last primitive plan ELMER is known to have been in to see if windows attached to these nearby plans match what is seen at the current location. If not, then the hierarchical structure of a plan can be useful since it can be viewed as a plan traversing ever larger regions as you move up the hierarchy. Presumably ELMER is still within a region traversed at some level by the current plan (unless he has gone very wrong indeed), and this information can be used to avoid searching the Map's entire repetoire of windows. Using this intuition, it is possible to move up the hierarchy from the primitive level to a more abstract levels. Associations/inassociations can be taken at the higher levels and all windows attached to primitive descendents of such associated/inassociated routes can be similiarly matched to the current location for recognizable features. The process can continue until no further abstraction is

possible or until a recognizable location is found. In the latter case, a plan to get from the primitive route containing the matching window to the destination can be readily constructed.

In the former case, the current location is simply not in the Map (even by inference) so all that is left is to try to **explore** for some recognizable location. Exploration needs a direction to explore in and a set of termination conditions to stop exploration. Both can be obtained from the last known location in the current plan. The direction is merely the direction of the destination relative to the last known location. Presumably (although not always) ELMER hasn't gone far enough off course to alter this relative direction. The termination conditions are obtained by taking all transitions from routes near the last known location (ie. the routes generated above although the abstraction process can be stopped somewhat earlier if a smaller radius of exploration is desired). The exploration phase then proceeds with ELMER heading in the direction indicated (insofar as this is possible) until one of the transition labels matches, indicating that he is back in "known territory". Re-planning can then occur from that location.

The ability to explore turns out to be useful in other situations as well, in particular planning. If the Planner is unable to splice together two routes to form a plan, then it can stick an **exploration sub-plan** in to bridge the gap provided it knows the relative direction of the sub-routes being so bridged. This is non-trivial, unfortunately, unless the Map is extended to have some sort of global co-ordinate system. Such an extension is being designed, as are other Map extensions to allow the inference of various kinds of hypothetical routes based on categorizing various areas of the city as *grid* or *crescent* or the like (as is done in Kuipers [1977]). But a discussion of these aspects is beyond the scope of this paper.

4. CONCLUSION

In conclusion this paper illustrates the usefulness of an integrated view to the problem of recovering from execution errors. The uniform structure of routes and plans is important when trying to find ELMER's location or terminate an exploration. It is possible to associate/inassociate to nearby routes from the current plan which when combined with the hierarchical structure allows a focussing on relevant Map routes. Hierarchical plan structure (absent for example from Srinivas' [1977] system) is thus useful in error handling.

The close co-operation of the Planner, Executor and Map is also useful. The Planner helps the Executor by providing explicit error transitions to help the Executor determine when errors have occurred. The Executor helps the Planner by being able to retrace steps and explore without needing re-planning; it also helps out by being able to execute exploration sub-plans to bridge unplannable gaps. But the main interaction occurs between the Executor and the Map where the Map's summaries learned from previous experiences, prove invaluable. The Map provides window information to the Executor to help execution monitoring and to help determine ELMER's location once he gets lost. It also provides summaries of previous execution errors in order that the Executor can explicitly avoid these in the future.

Apart from the occasional resort to domain specific heuristics (eg. in judging the relevence of window information) most of the error recovery techniques are not restricted to the geographic microworld. We are currently exploring other applications to test the generality of these techniques.

5. ACKNOWLEDGMENTS

We would like to acknowledge the financial support of the National Sciences and Engineering Research Council of Canada and the University of Saskatchewan.

6. REFERENCES

[1] Hayes-Roth, B. and Hayes-Roth, F. (1979). A Cognitive Model of Planning. *Cognitive Science 3*, October-December,pp. 275-310.

[2] Kuipers, B.J. (1977). Representing Knowledge of Large Scale Space. AI Lab. *AI-TR-418*, MIT, Cambridge, Mass.

[3] McCalla, G.I., Schneider, P.F., Cohen, R. & Levesque, H. (1978). Investigations into Planning and Executing in an Independent and Continuously Changing Microworld. *AI Memo 78-2*, Department of Computer Science, University of Toronto, Ontario.

[4] McCalla, G.I. & Schneider, P.F. (1979). The Execution of Plans in an Independent Dynamic Microworld. *Proceedings: Sixth International Joint Conference of Artificial Intelligence*, Tokyo, Japan.

[5] McCalla, G.I., Reid, L. & Schneider, P.F. (1982). Plan Creation, Plan Execution and Knowledge Acquisition in a Dynamic Microworld. *International Journal of Man-Machine Studies 16*, pp. 89-112.

[6] Robinson, A.E. & Wilkins, D.E. (1980). Representing Knowledge in an Interactive Planner. *Proceedings: First Annual National Conference on Artificial Intelligence*, Stanford, California.

[7] Srinivas, S. (1977). *Error Recovery in Robot Systems*, CIT, Pasadena, California.

[8] Sussman, G.J. (1973). A Computational Model of Skill Acquisition. AI Lab. *AI-TR-297*, MIT, Cambridge, Mass.

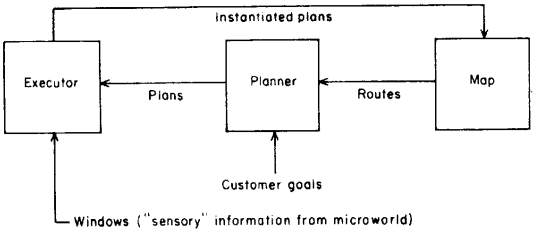

FIG. 1. Basic system architecture.

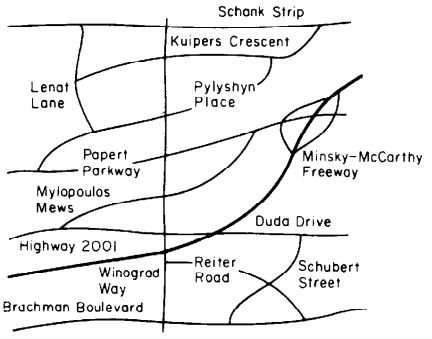

FIG. 2. Simon City.

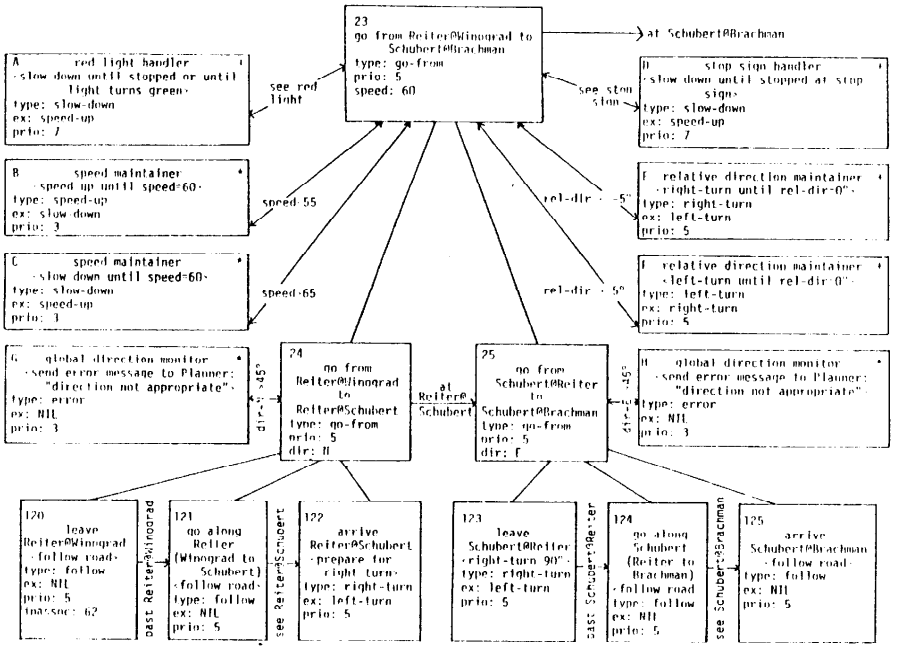

Figure 3 - A Typical Plan

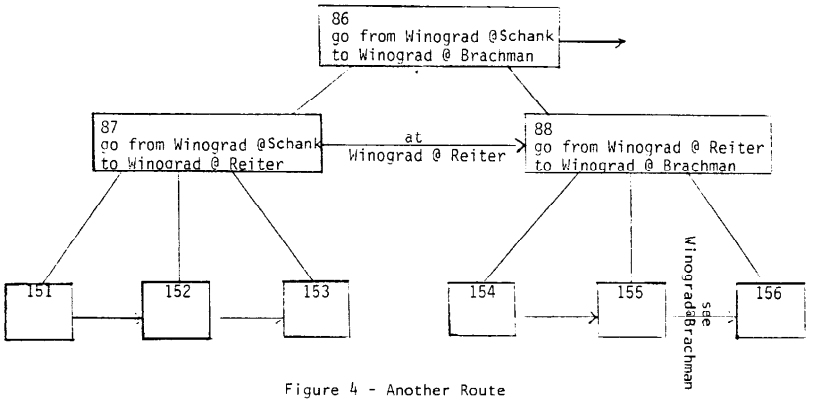

Figure 4 - Another Route

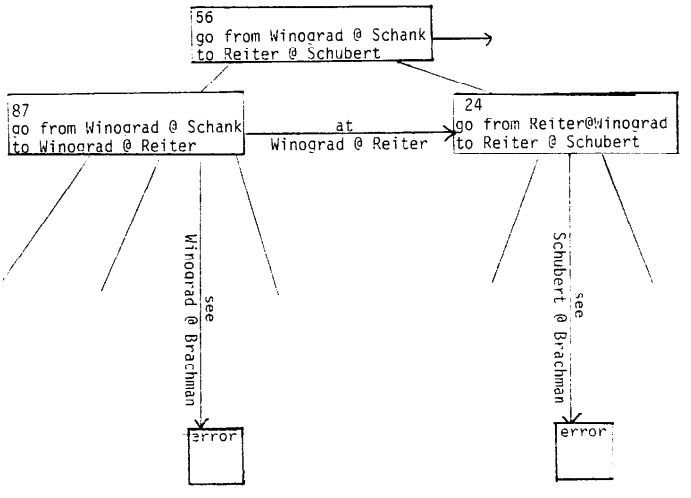

Figure 5 - Explicit Error Transitions