In Proceedings of the First International Conference on Autonomous Agents, Feb. 1997, Menlo Park, CA

High-Level Planning and Low-Level Execution: Towards a Complete Robotic Agent *

Karen Zita Haigh
khaigh@cs.cmu.edu
http://www.cs.cmu.edu/~khaigh

Manuela M. Veloso
mmv@cs.cmu.edu
http://www.cs.cmu.edu/~mmv

Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213-3891

Abstract

We have been developing ROGUE, an architecture that integrates high-level planning with a low-level executing robotic agent. ROGUE is designed as the office gofer task planner for Xavier the robot. User requests are interpreted as high-level planning goals, such as getting coffee, and picking up and delivering mail or faxes. Users post tasks asynchronously and ROGUE controls the corresponding planning and execution continuous process. This paper presents the extensions to a nonlinear state-space planning algorithm to allow for the interaction to the robot executor. We focus on presenting how executable steps are identified based on the planning model and the predicted execution performance; how interrupts from users requests are handled and incorporated into the system; how executable plans are merged according to their priorities; and how monitoring execution can add more perception knowledge to the planning and possible needed re-planning processes. The complete ROGUE system will learn from its planning and execution experiences to improve upon its own behaviour with time. We finalize the paper by briefly discussing ROGUE's learning opportunities.

*This research is sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Material Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the Wright Laboratory or the U. S. Government.

1 Introduction

We have been working towards the goal of building autonomous robotic agents that are capable of planning and executing high-level tasks. Our framework consists of the integration of Xavier the robot [10; 14] with the PRODIGY planning system [3; 17] in a setup where users can post tasks for which the planner generates appropriate plans, delivers them to the robot, and monitors their execution. ROGUE effectively acts as the task scheduler for the robot. Currently, ROGUE has the following capabilities: (1) a system that can generate and execute plans for multiple interacting goals which arrive asynchronously and whose task structure is not known *a priori*, interrupting and suspending tasks when necessary, and (2) a system which can compensate for minor problems in its domain knowledge, monitoring execution to determine when actions did not achieve expected results, and replanning to correct failures.

One of the goals of the project is to have the robot move autonomously in an office building reliably performing office tasks such as picking up and delivering mail and computer printouts, returning and picking up library books, and carrying recycling cans to the appropriate containers [12]. Our on-going contribution to this ultimate goal is at the high-level reasoning of the process, allowing the robot to efficiently handle multiple interacting goals, and to learn from its experience. We aim at building a complete planning, executing and learning autonomous robotic agent.

We have developed techniques for the robot to autonomously perform many-step plans, and to appropriately handle asynchronous user interruptions with new task requests. We are currently investigating techniques that will allow the system to use experience to improve its performance and model of the world.

We have been reporting on our work on the interleaving of planning and execution work [7; 8]. In this paper, we focus on describing in detail the planning algorithm and representation. ROGUE uses the PRODIGY planning algorithm which is a non-linear state-space means-ends analysis planner. We explain the extensions to the algorithm that allow for effective robot execution. We describe how the planning algorithm is biased towards the identification of potentially executable steps and can be interrupted asynchronously with new goal requests. The planner communicates with the robot executor and information perceived from execution is converted to planning information. Re-planning may take into account information gathered from execution. The paper presents the features of the current algorithm as well as our on-going research work to extend the current features not only along more sophisticated planning and execution representations, but also along learning from execution.

The paper is organized as follows: In Section 2 we introduce the ROGUE architecture, our developed integrated system. In Section 3, we give a brief introduction to the PRODIGY planner. In Section 4, we describe how PRODIGY's means-ends engine incorporates multiple goals. We present the mechanism used to translate from symbolic actions to real world execution in Section 5. We describe the behaviour of the architecture in a dynamic environment in Section 6. We present a brief review of related work in Section 7. Finally we provide a summary of ROGUE's current capabilities in Section 8 along with a description of our future work to incorporate learning methods into the system.

2 General Architecture

ROGUE[1] is the system built on top of PRODIGY4.0 to communicate with and to control the high-level task planning in Xavier[2]. The system allows users to post tasks for which the planner generates a plan, delivers it to the robot, and then monitors its execution. ROGUE is intended to be the task scheduler for a roving office gofer unit, and will deal with tasks such as delivering mail, picking up printouts and returning library books.

Xavier is a mobile robot being developed at CMU [10; 14] (see Figure 1). It is built on an RWI B24 base and includes bump sensors, a laser range finder, sonars, a color camera and a speech board. The software controlling Xavier includes both reactive and deliberative behaviours, integrated using the Task Control Architecture (TCA) [13; 15]. The underlying architecture is described in more detail in [14].

PRODIGY and Xavier are linked together using the Task Control Architecture [13; 15] as shown in Figure 2. Currently, ROGUE's main features are (1) the ability to receive and reason about multiple asynchronous goals, suspending and interrupting actions when necessary, and (2) the ability to sense, reason about, and correct simple execution failures.

[1]In keeping with the Xavier theme, ROGUE is named after the "X-men" comic-book character who absorbs powers and experience from those around her. The connotation of a wandering beggar or vagrant is also appropriate.

[2]We will use the term Xavier when referring to features specific to the robot, PRODIGY to refer to features specific to the planner, and ROGUE to refer to features only seen in the combination.

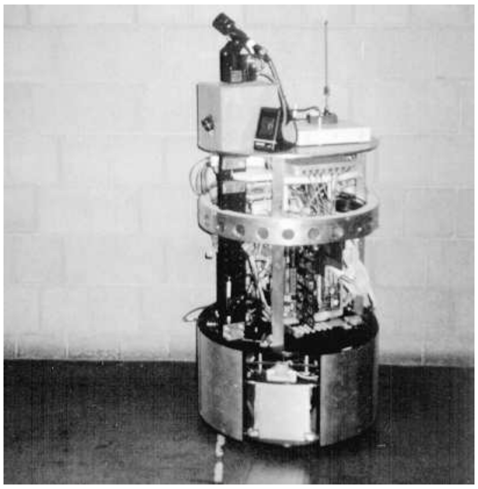

Figure 1: Xavier the Robot

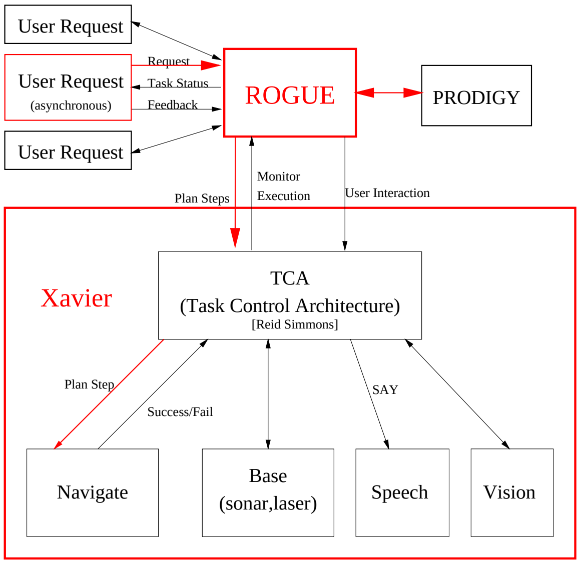

Figure 2: Rogue Architecture

3 Prodigy

ROGUE is designed to be used by multiple users in a dynamic environment. It therefore needs to have the capability to integrate new task requests into its planning structures as well as to handle and correct failures. PRODIGY's means-ends analysis search engine makes many of ROGUE's features easy to implement.

PRODIGY is a domain-independent problem solver that serves as a testbed for machine learning research [3; 17]. PRODIGY4.0 is a nonlinear planner that follows a state-space search guided by means-ends analysis and backward chaining. It reasons about multiple goals and multiple alternative operators to achieve the goals.

In PRODIGY, an incomplete plan consists of two parts, the *head-plan* and the *tail-plan* (see Figure 3). The

Back-Chainer
1. Pick an unachieved goal or precondition literal l.
   Decision point: Choose an unachieved literal.
2. Pick an operator *op* that achieves l.
   Decision point: Choose an operator that achieves this literal.
3. Add *op* to the tail-plan.
4. Instantiate the free variables of *op.*
   Decision point: Choose an instantiation for the variables of the operator.

Operator-Application
1. Pick an operator *op* in *Tail-Plan* which is an *applicable operator*, that is
   (A) there is no operator in *Tail-Plan* ordered before *op*, and
   (B) the preconditions of *op* are satisfied in the current state C.
   Decision point: Choose an applicable operator to apply.
2. Move *op* to the end of *Head-Plan* and update the current state C.

Prodigy
1. If the goal statement G is satisfied in the current state C, then return *Head-Plan.*
2. Either (A) *Back-Chainer* adds an operator to the *Tail-Plan*, or
   (B) *Operator-Application* moves an operator from *Tail-Plan* to *Head-Plan.*
   Decision point: Decide whether to apply an operator or to add an operator to the tail.
3. Recursively call *Prodigy* on the resulting plan.

Table 1: Prodigy decision points.

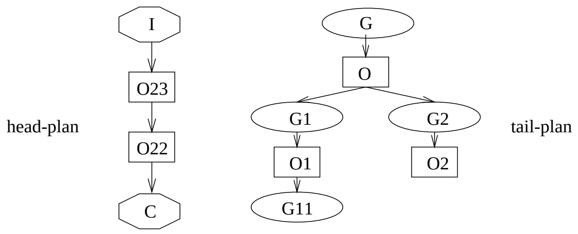

Figure 3: Representation of an incomplete plan.

tail-plan is built by the partial-order backward-chaining algorithm, which starts from the goal statement G and adds operators, one by one, to achieve preconditions of other operators that are untrue in the current state, *i.e. pending goals.* The head-plan is a *valid total-order plan*, that is, a sequence of operators that can be applied to the initial state I.

The planning reasoning cycle involves several decision points, including which goal to select from the set of pending goals, and which applicable action to execute (*i.e.* move from the tail-plan to the head-plan). There may be several different ways to achieve a goal, but the choices about which action to take are made while expanding the tail-plan, and only one of those choices is executed. Table 1 shows the decisions made while creating the plans. **Back-Chainer** shows the decisions made while back-chaining on the tail-plan, **Operator-Application** shows how the operator is added to the head-plan, and **Prodigy** shows the mediation step.

Prodigy provides a method for creating *search control rules* which reduces the number of choices at each decision point by pruning the search space or suggesting a course of action while expanding the tail-plan. In particular, control rules can select, prefer or reject a particular goal or action in a particular situation. Control rules can be used to focus planning on particular goals and towards desirable plans. Dynamic goal selection from the set of pending goals enables the planner to interleave plans, exploiting common subgoals and addressing issues of resource contention.

Prodigy maintains an internal model of the world in which it simulates the effects of selected applicable operators. The state C achieved by applying the head-plan to the initial state is called the *current state.* The back-chaining algorithm responsible for the tail-plan views C as its initial state. Applying an operator can therefore give the planner additional information (such as consumption of resources) that might not be accurately predictable from the domain model.

Prodigy also supports real-world execution of its applicable operators when it is desirable to know the actual outcome of an action; for example, when actions have probabilistic outcomes, or the domain model is incomplete and it is necessary to acquire additional knowledge for planning. During the application phase, user-defined code is called which can map the operator to a real-world action sequence [16]. Some examples of the use of this feature include shortening combined planning and execution time, acquiring necessary domain knowledge in order to continue planning (*e.g.* sensing the world), and executing an action in order to know its outcome and handle any failures.

At each PRODIGY decision point
(control-rule SELECT-TOP-PRIORITY-AND-COMPATIBLE-GOALS
(if (and (candidate-goal <*goal*>)
(or (ancestor-is-top-priority-goal <*goal*>)
(compatible-with-top-priority-goal <*goal*>))))
(then select goal <*goal*>))

Table 2: Goal selection search control rule

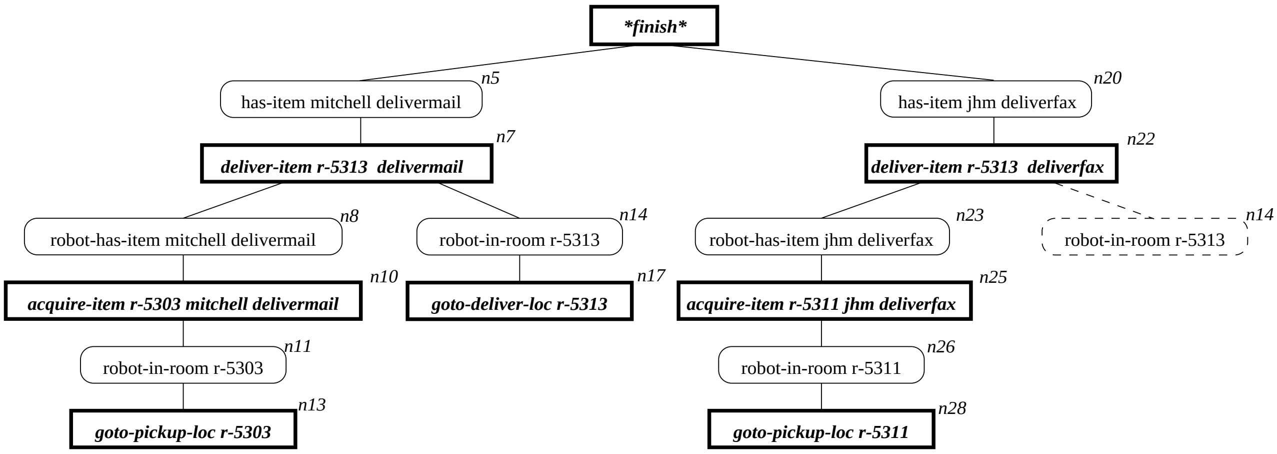

Figure 4: Search Tree for two task problem; goal nodes in ovals, required actions in rectangles.

4 Handling Asynchronous Requests

In the general case, while ROGUE is executing the plan to achieve some goal, other users may submit goal requests. ROGUE does not know *a priori* what these requests will entail. One common method for handling these multiple goal requests is simply to process them in a first-come-first-served manner; however this method ignores the possibility that new goals may be more important or could be achieved opportunistically.

ROGUE has the ability to process incoming asynchronous goal requests, prioritize them and identify when different goals could be achieved opportunistically. It is able to temporarily suspend lower priority actions, resuming them when the opportunity arises; and it is able to successfully interleave compatible requests.

When a new request comes in, ROGUE adds it to PRODIGY's pending goals cache and updates the domain model. When PRODIGY reaches the next decision point, it fires any relevant search control rules; it is at this point when PRODIGY first starts to reason about the newly added task request.

Search control rules force the planner to focus its planning effort on selected or preferred goals, as described above. Table 2 shows ROGUE's goal selection control rule which calls two functions, forcing PRODIGY to select those goals with high priority along with those goals which can be opportunistically achieved without compromising the main high-priority goal.

Once PRODIGY has selected its set of immediate goals, it expands their tail-plans in the normal means-ends analysis manner. The tail-plans for each of the suspended tasks remain unaffected. The control rule feature of PRODIGY permits plans and actions for one goal to be interrupted by another without necessarily affecting the validity of the planning for the interrupted goals. PRODIGY simply suspends the planning for the interrupted goal, plans for and achieves the selected goal, and then returns to planning for the interrupted goal.

Note that while PRODIGY is expanding the tail-plan for a particular selected goal, it is still making global decisions and may decide to change focus again. PRODIGY's means-ends search engine supports dynamic goal selection and changing objectives by making it easy to suspend and reactivate tasks.

The tail-plan shown in Figure 4 shows how PRODIGY expands the two goals `(has-item mitchell delivermail)` and `(has-item jhm deliverfax)`. This complete tail-plan would have been generated if no steps had been moved to the head-plan yet, *i.e.* if no execution had started. The tail-plan to achieve both requests can be viewed as two separate head-plans, as shown in Figure 5.

To find a good execution order of these applicable actions, ROGUE selects the one that minimizes the expected total traveled distance from the current location. This choice is an execution-driven heuristic to effectively merge these two head-plans.

```
Head-plan 1:
  <goto-pickup-loc mitchell r-5303>
  <acquire-item r-5303 mitchell delivermail>
  <goto-deliver-loc mitchell r-5313>
  <deliver-item r-5313 mitchell delivermail>

Head-plan 2:
  <goto-pickup-loc jhm r-5311>
  <acquire-item r-5311 jhm deliverfax>
  <goto-deliver-loc jhm r-5313>
  <deliver-item r-5313 jhm deliverfax>
```

Figure 5: Two executable plans to be merged for the tail-plan in Figure 4.

(Note that incompatible actions are not among the choices and their tail-plans will be expanded later.) In this situation, Rogue finds the shortest route that achieves both tasks. Actions that opportunistically achieve goals of other tasks are not repeated, *e.g.* both `<goto-deliver-loc jhm r-5313>` and `<goto-deliver-loc mitchell r-5313>` achieve the same goal, namely `(robot-in-room r-5313)`, so therefore only one of the actions will be executed.

Note however that the most common situation is that requests arrive asynchronously, and thus part of a complete tail-plan for a specific goal may have already been moved to the head-plan and therefore executed. For example, the second request (`jhm`) in Figure 4 may have arrived after the first had already been partially executed. Instead of merging all steps of all plans, Rogue must merge the steps for the new request with the *remaining* steps of the partially executed plan. Figure 6 shows one possible execution sequence.

The complete procedure for achieving a particular task is summarized as follows:

1. Receive task request.
2. Add knowledge to state model, create top-level goal.
3. Create tail-plan.
4. Move actions to the head-plan, sending execution commands to robot, and monitoring outcome.

Separate tail-plans are created for each request, and their head-plans are merged into a single execution sequence, *i.e.* only step 4 changes, so that the actions moved to the head-plan are selected from amongst the complete set of existing tail-plans.

```
Solution:
      <goto-pickup-loc mitchell r-5303>
   [arrival of second request]
      <acquire-item r-5303 mitchell delivermail>
      <goto-pickup-loc jhm r-5311>
      <acquire-item r-5311 jhm deliverfax>
      <goto-deliver-loc mitchell r-5313>
      <deliver-item r-5313 jhm deliverfax>
      <deliver-item r-5313 mitchell delivermail>
```

Figure 6: Final Execution Sequence

5 Symbolism to Realism

In this section we describe the interaction between the planner and the robot, showing how symbolic action descriptions are turned into robot commands, as well as how deliberate observation is used by the system to make intelligent planning decisions.

The key to this communication model is based on a pre-defined language and model translation between PRODIGY and Xavier. PRODIGY relies on a state description of the world to plan. Rogue is capable of converting Xavier's actual perception information into PRODIGY's state representation, and Rogue's monitoring algorithm determines which information is relevant for planning and replanning. Similarly Rogue is capable of translating plan steps into Xavier's actions commands.

When PRODIGY moves a plan step from the tail-plan to the head-plan, Rogue translates the high-level abstract action into a command sequence appropriate for execution. The action `acquire-item`, for example is mapped to a sequence of commands that allows the robot to interact with a human. For example, the action <`GOTO-LOCATION ROOM`>) is mapped to the commands (1) find the coordinates of the room, and (2) navigate to those coordinates.

```
SENDING COMMAND:
   (TCAEXPANDGOAL "navigateToG"
             #(MAP-DATA 567.0d0 2316.5d0))
```

These command sequences are manually generated but incremental in nature. They may be executed directly by the Rogue module (*e.g.* an action like `finger`), or sent via the TCA interface to the Xavier module designed to handle the command.

Figure 7 shows a partial trace of a run. When PRODIGY applies the <`GOTO-ROOM`> operator in its internal world model (see node `n14`), Rogue sends the command to Xavier for execution. Each line marked "`SENDING COMMAND`" indicates a direct command sent through the TCA interface to one of Xavier's modules.

This example shows the use of two more TCA commands, namely `C_observe` and `C_say` (after nodes `n14` and `n18`). The first command is a direct perception action. The observation routine can vary depending on the kind of information needed. It can range from an actual interpretation of some of Xavier's sensors or its visual images, to specific input by a user. The command `C_say` sends the string to the speech board.

Linking a symbolic planner to a robot executor requires not only that the planner is capable generating partial plans for execution in a continuous way, but that the dynamic nature of the real world can be captured in the planners' knowledge base. The planner must continuously re-evaluate the goals to be achieved based on current state information. Rogue enables this link by both mapping PRODIGY's plan steps into Xavier's commands and by abstracting Xavier's perception information PRODIGY's state information.

```
    n2 (done)
    n4 <*finish*>
      n5 (mtg-scheduled)
Firing prefer bindings LOOK-AT-CLOSEST-CONF-ROOM-FIRST #<5309> over #<5311>
      n7 <schedule-meeting 5309> [1]
        n8 (conference-room 5309)
        n10 <select-conference-room 5309>
          n11 (at-room 5309)
          n13 <goto-room 5309>
          n14 <GOTO-ROOM 5309>

SENDING COMMAND (tcaExecuteCommand "C_say" "Going to room 5309")
ANNOUNCING: Going to room 5309
SENDING COMMAND (TCAEXPANDGOAL "navigateToG" #(TASK-CONTROL::MAPLOCDATA 567.0d0 3483.0d0))
   ...waiting...
Action NAVIGATE-TO-GOAL finished (SUCCESS).

          n15 (room-empty 5309)
          n17 <observe-conference-room 5309>
          n18 <OBSERVE-CONFERENCE-ROOM 5309>

SENDING COMMAND (tcaExecuteCommand "C_observe" "5309")
DOING OBSERVE: Room 5309 conf-room
   ...waiting...
Action OBSERVE finished (OCCUPIED).

SENDING COMMAND (tcaExecuteCommand "C_say" "This room is occupied")
ANNOUNCING: This room is occupied

  6   n6 schedule-meeting
  7   n15 <schedule-meeting r-5311>
```

Figure 7: Trace of Rogue interaction.

6 Monitoring Execution, Detecting Failures & Replanning

The capabilities described in the preceding section are sufficient to create and execute a simple plan in a world where all dynamism is predictable. The real world, however, needs a more flexible system that can monitor its own execution and compensate for problems and failures. Any action that is executed by any agent is not guaranteed to succeed in the real world.

There are several approaches to reason about the real world uncertainty. In one extreme, planners can completely ignore that the world is uncertain. They follow a deterministic model and generate a single executable plan. When execution failures occur, replanning is invoked. At the other extreme, conditional planning aims at considering in the domain model all the possible contingencies of the world and plan ahead for each individual one. Because it is impossible to enumerate all the world's possible events, complete conditional planning is infeasible. Probabilistic planning falls in the middle of these two frameworks [1]. At planning time, it accounts for the most probable contingencies and relies on replanning if unpredictable or rare events may take place and disrupt the plan execution.

Probabilistic planners may increase the probability of a plan succeeding, but the domain model underlying the plan is bound to be incompletely or incorrectly specified. Not only is the world more complex than a model, but it is also constantly changing in ways that cannot be predicted. Therefore every agent executing in the real world must have the ability to monitor the execution of its actions, detect when the actions fail, and compensate for these problems.

The TCA architecture provides mechanisms for monitoring the progress of actions. Rogue currently monitors the outcome of the `navigateToG` command. Since the navigate module may get confused and report a success even in a failure situation, Rogue always verifies the location with a secondary test (vision or human interaction). If Rogue detects that in fact the robot is *not* at the correct goal location, Rogue updates prodigy's domain knowledge to reflect the actual position, rather than the expected position.

This update has the direct effect of indicating to prodigy that the execution of an action failed, and it will backtrack to find a different action which can achieve the goal. Since PRODIGY's search algorithm is state-based, it examines the current state before making each decision. If the preconditions for a given desirable action are not true, PRODIGY must attempt to achieve them. Therefore, when an action fails, the *actual* outcome of the action is not the same as the *expected* outcome, and PRODIGY will attempt to find another solution.

In a similar manner, PRODIGY is able to detect when an action is no longer necessary. If an action unexpectedly achieves some other necessary part of the plan, then that precondition is added to the state and PRODIGY will not need to subgoal to achieve it

Also, when an action accidentally *dis*achieves the effect of a previous action (and the change is detectable), ROGUE deletes the relevant precondition and PRODIGY will be forced to reachieve it.

Take for example, a coffee delivery scenario. The system picks up the coffee, adding the literal `(has-item coffee)` to its knowledge base and deleting the goal `(pickup-item coffee roomA)`. If ROGUE is now interrupted with a more important task, it suspends the coffee delivery and does the other task. While doing the new task, the coffee gets cold, making the literal `(has-item coffee)` untrue. (The state change is detected by a manually encoded daemon.) When PRODIGY returns to the original task, it examines the next foreseeable action: `(deliver-item coffee roomB)`, discovers that a precondition is missing (it doesn't have the coffee) and will subgoal on re-achieving it.

In this manner, ROGUE is able to detect simple execution failures and compensate for them. The interleaving of planning and execution reduces the need for replanning during the execution phase and increases the likelihood of overall plan success. It allows the system to adapt to a changing environment where failures can occur.

Observing the real world allows the system to adapt to its environment and to make intelligent and relevant planning decisions. Observation allows the planner to update and correct its domain model when it notice changes in the environment. For example, it can notice limited resources (*e.g. battery*), notice external events (*e.g. doors opening/closing*), or prune alternative outcomes of an operator. In these ways, observation can create opportunities for the planner and it can also reduce the planning effort by pruning possibilities. Real-world observation creates a more robust planner that is sensitive to its environment.

7 Related Work

Following is a brief description of some of the robot architectures most similar to ROGUE, pointing out some of the major differences.

Shakey [9] was the first system to actually use plans to control a real robot in tasks involving pushing boxes. It also had a limited ability to reuse successful plans. The robot had a simple vision system and could identify failures and plan to correct them. Shakey however operated in a very simple near-static world doing very simple single-goal tasks. The range of failures that could occur were very limited, and goals were not very challenging. There was little need for complex high-level reasoning or learning.

PARETO [11], can plan to acquire information and recognize opportunities in the environment (as can ROGUE), but relies on powerful, perfect sensing in a simulated world. Plans for PARETO are created up-front by a contingency planner which needs to reason about all possible failures, whereas ROGUE exploits interleaved planning and execution and uses the actual outcome of an action to decide which branch of the plan to expand, not only reducing the total planning effort, but also the modelling effort since not all failures need to be predicted.

ATLANTIS [5] and RAP [4], like TCA, are architectures that enable a library of behaviours and reactions to be controlled by a deliberative system. The have been used as the underlying control mechanism on a variety of robots, from indoor mobile robots [5] to space station repair robots [2] and unmanned spacecraft [6]. We believe that ROGUE is the only such system that can support asynchronous goals, but since each of these architectures is inherently extensible, the behaviours demonstrated by ROGUE under TCA could be easily transferred to one of the other architectures.

8 Summary

In this paper we have presented one aspect of ROGUE, an integrated planning and execution robot architecture. We have described here how PRODIGY's state-space means-ends planning algorithm gives ROGUE the power

- to easily integrate asynchronous requests,
- to prioritize goals,
- to easily suspend and reactivate tasks,
- to recognize compatible tasks and opportunistically achieve them,
- to execute actions in the real world, integrating new knowledge which may help planning, and
- to monitor and recover from failure.

ROGUE represents a successful integration of a classical AI planner with a real mobile robot. The complete planning & execution cycle for a given task can be summarized as follows:

1. ROGUE requests a plan from PRODIGY.
2. PRODIGY passes executable steps to ROGUE.
3. ROGUE translates and sends the planning steps to Xavier.
4. ROGUE monitors execution and through observation identifies goal status; failure means that PRODIGY's domain model is modified and PRODIGY may backtrack or replan for decisions

As described here, ROGUE is fully implemented and operational. The system completes all requested tasks, running errands between offices in our building. In the period from December 1, 1995 to May 31, 1996 Xavier attempted 1571 navigation requests and reached its intended destination in 1467 cases, where each job required it to move 40 meters on average for a total travel distance of over 60 kilometers. Results are presented in greater detail elsewhere in these proceedings [14].

This work is the basis for machine learning research with the goal of creating a complete agent that can reliably perform tasks that it is given. Learning allows the agent to use accumulated experience and feedback about its performance to improve its behaviour. Without learning, the behaviour of an autonomous agent is completely dependent on the predictive ability of the programmer.

We intend to implement learning behaviour to notice patterns in the environment so that failures can be predicted and avoided. We would like, for example, to be able to say *"At noon I avoid the lounge"*, or *"That task can only be completed after 10am"*, or even something as apparently simple as *"I can't do that task given what else I have to do."* Learning would occur at three levels:

- during navigation to select appropriate routes,
- during single-task planning to place constraints on when it can be done, and
- during multiple-task planning to place constraints on when tasks can be successfully combined.

When complete, ROGUE will learn from real world execution experience to improve its high-level reasoning capabilities.

References

[1] Jim Blythe. Planning with external events. In Ramon Lopez de Mantaras and David Poole, editors, *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 94–101, Seattle, WA, July 1994. Morgan Kaufmann.

[2] R. Peter Bonasso and David Kortenkamp. Using a layered control architecture to alleviate planning with incomplete information. In *Proceedings of the AAAI Spring Symposium "Planning with Incomplete Information for Robot Problems"*, pages 1–4, Menlo Park, CA, March 1996. AAAI Press.

[3] Jaime G. Carbonell, Craig A. Knoblock, and Steven Minton. PRODIGY: An integrated architecture for planning and learning. In K. VanLehn, editor, *Architectures for Intelligence*. Erlbaum, Hillsdale, NJ, 1990. Also Available as Technical Report CMU-CS-89-189.

[4] R. James Firby. Task networks for controlling continuous processes. In K. Hammond, editor, *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS-94)*, pages 49–54, Menlo Park, CA, June 1994. AAAI Press.

[5] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 809–815, 1992.

[6] Erann Gat. News from the trenches: An overview of unmanned spacecraft for AI researchers. In *Proceedings of the AAAI Spring Symposium "Planning with Incomplete Information for Robot Problems"*, pages 5–12, Menlo Park, CA, March 1996. AAAI Press.

[7] Karen Zita Haigh and Manuela Veloso. Interleaving planning and robot execution for asynchronous user requests. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*, November 1996. To Appear.

[8] Karen Zita Haigh and Manuela M. Veloso. Using perception information for robot planning and execution. In *Proceedings of the AAAI Workshop "Intelligent Adaptive Agents"*, pages 23–32, Menlo Park, CA, August 1996. AAAI Press. Available at `http://www.cs.cmu.edu/~khaigh/papers.html`.

[9] Nils J. Nilsson. Shakey the robot. Technical Report 323, AI Center, SRI International, Menlo Park, CA, 1984.

[10] Joseph O'Sullivan and Karen Zita Haigh. *Xavier*. Carnegie Mellon University, Pittsburgh, PA, July 1994. Manual, Version 0.2, unpublished internal report. Available via `http://www.cs.cmu.edu/~Xavier/`.

[11] Louise Margaret Pryor. *Opportunities and Planning in an Unpredictable World*. PhD thesis, Northwestern University, Evanston, Illinois, 1994. Also available as Technical Report number 53.

[12] Reid Simmons. Becoming increasingly reliable. In K. Hammond, editor, *Artificial Intelligence Planning Systems: Proceedings of the Second International Conference (AIPS-94)*, pages 152–157, Menlo Park, CA, June 1994. AAAI Press.

[13] Reid Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1):34–43, February 1994.

[14] Reid Simmons, Rich Goodwin, Karen Zita Haigh, Sven Koenig, and Joseph O'Sullivan. A modular architecture for office delivery robots. In *The First International Conference on Autonomous Agents*, February 1997.

[15] Reid Simmons, Long-Ji Lin, and Chris Fedor. Autonomous task control for mobile robots. In *Proceedings of the IEEE Symposium on Reactive Control*, pages 663–668, Philadelphia, PA, September 1990.

[16] Peter Stone and Manuela Veloso. User-guided interleaving of planning and execution. In *Proceedings of the European Workshop on Planning*, pages 103–112, September 1995.

[17] Manuela M. Veloso, Jaime Carbonell, M. Alicia Pérez, Daniel Borrajo, Eugene Fink, and Jim Blythe. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 7(1):81–120, January 1995.