Interleaving Planning and Robot Execution for Asynchronous User Requests

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

This paper describes ROGUE, *an integrated planning and executing robotic agent.* ROGUE *is designed to be a roving office gopher unit, doing tasks such as picking up & delivering mail and returning & picking up library books, in a setup where users can post tasks for the robot to do. We have been working towards the goal of building a completely autonomous agent which can learn from its experiences and improve upon its own behaviour with time. This paper describes what we have achieved to-date: (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.*

1 Introduction

We have been working towards the goal of building an autonomous robot that is capable of planning and executing high-level tasks in a dynamic environment. To achieve this end, we have been building an integrated framework, ROGUE, which combines PRODIGY, a planning and learning system [17], with Xavier, an autonomous robot [9]. The setup allows users to post tasks for which PRODIGY generates appropriate plans, delivers them to Xavier, and monitors their execution. ROGUE acts as the task scheduler for the robot.

Xavier is a robot developed by Reid Simmons at Carnegie Mellon [9, 13]. One of the goals of the Xavier 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. 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. 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.

Other researchers investigate the problem of interleaving planning and execution (including [1, 4, 6, 7]). We build upon this work and pursue our investigation from three particular angles: that of real execution in an autonomous agent, in addition to simulated execution, that of challenging the robot with multiple asynchronous user-defined interacting tasks, and that of interspersing execution and replanning as an additional learning experience.

In this paper, we focus on presenting our current work on the interleaving of planning and execution by a real robot within a framework with the following sources of incomplete information:

- the tasks requested by the users are not completely specified,
- the set of all the goals to be achieved is not known *a priori*,
- the domain knowledge is incompletely or incorrectly specified, and
- the execution steps sent to the robot may not be achieved as predicted.

The learning portions of the system is the focus of our current work and will be the topic of future papers.

The paper is organized as follows: In Section 2 we introduce the ROGUE architecture, our developed integrated system. We illustrate the behaviour of ROGUE for a single goal when no errors occur during execution in Section 3. We describe the behaviour of the architecture with multiple goals and simple execution errors in Section 4. In Section 5 we briefly present

some related work. Finally we provide a summary of ROGUE's current capabilities in Section 5 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 a roving office gofer unit, and will deal with tasks such as delivering mail, picking up printouts and returning library books.

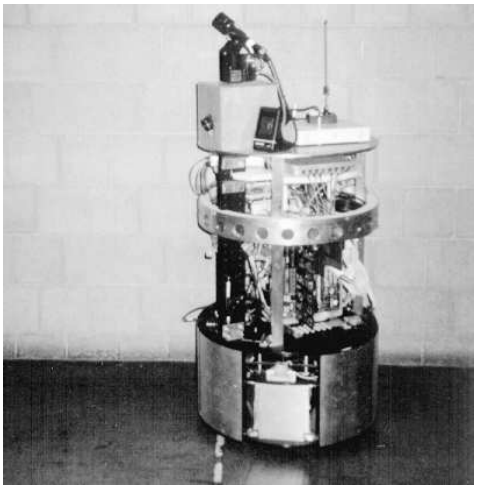


Figure 1: Xavier the robot

Xavier is a mobile robot being developed at CMU [9, 13] (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. Control, perception and navigation planning are carried out on two on-board Intel 80486-based machines. Xavier can communicate with humans via an on-board lap-top computer or via a natural language interface.

Beyond its research abilities, Xavier can autonomously perform one of a number of simple tasks for users via it's on-line WWW page: `http://www.cs.cmu.edu/~Xavier`. To date, Xavier has been operational more than 180 hours, covering almost 60km and completing more than 90% of its tasks.

The software controlling Xavier includes both reactive and deliberative behaviours, integrated using the Task Control Architecture (TCA) [12, 14]. TCA provides facilities for scheduling and synchronizing tasks, resource allocation, environment monitoring and exception handling. The reactive behaviours enable the robot to handle real-time local navigation, obstacle avoidance, and emergency situations (such as detecting a bump). The deliberative behaviours include vision interpretation, maintenance of occupancy grids & topological maps, and path planning & global navigation (an A* algorithm).

All modules and behaviours operate independently, concurrently and in a distributed manner; they can also be modified or added incrementally without affecting existing behaviours. The clear separation between reactive and deliberative behaviours increases system predictability by isolating different concerns: the robot's behaviour during normal operation is readily apparent, while strategies for handling exceptions can be individually analyzed.

PRODIGY and Xavier are linked together using the Task Control Architecture [12, 14] 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 reason about and correct simple execution failures.

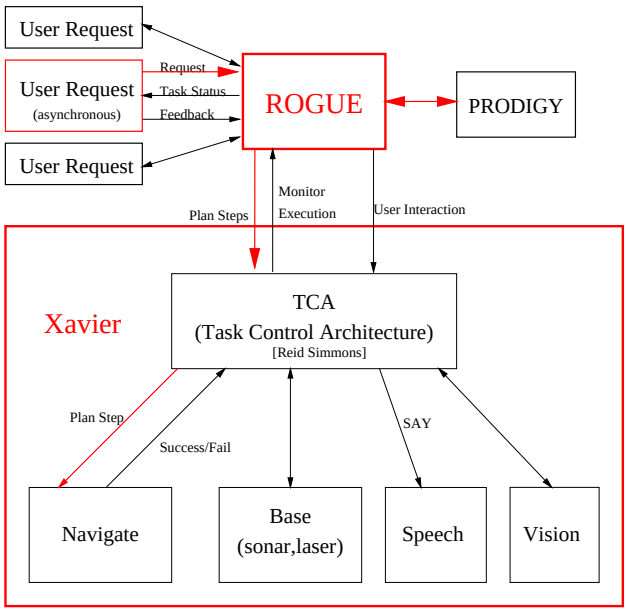


Figure 2: Rogue Architecture

2.1 PRODIGY

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 uses means-ends analysis and backward chaining to reason about multiple goals and multiple alternative operators to achieve the goals.

The planning reasoning cycle involves several decision points, including which goal to select from the

[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.

set of pending goals, and which applicable action to execute.

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. 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. Applying an operator gives 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 absolutely necessary to know the 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. During the application phase, user-defined code is called which can map the operator to a real-world action sequence [15]. 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.

3 Base-line Behaviour

This section describes ROGUE's underlying architecture in more detail, describing the interface for users to create task requests, and then, through the use of an example, describes how the planner generates a plan to achieve the request and executes it, successfully making an office delivery. The features described here were developed using the Xavier simulator and then tested on the actual robot.

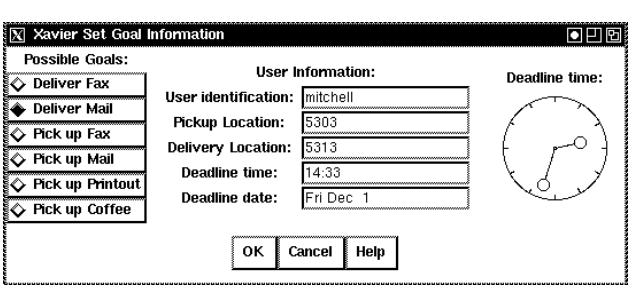
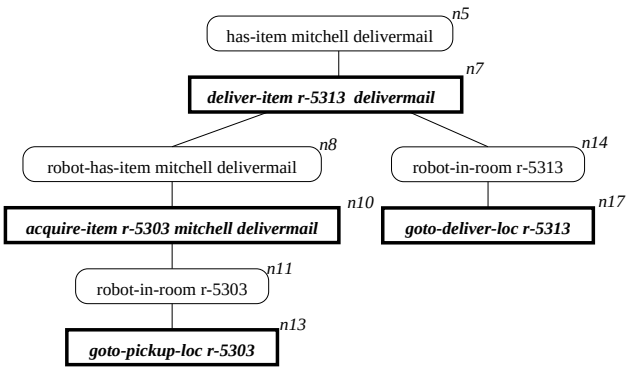


Figure 3: User Request Interface

Any user can create and send a goal request to ROGUE via a simple user interface, shown in Figure 3. These requests can come in asynchronously, and include information about what item needs to be moved, where it needs to be picked up, where it needs to be delivered, and who is making the request. ROGUE is able to identify and handle incomplete goal information by utilizing default values and accessing various on-line information sources (such as `finger`), or requesting them from the user.

Consider a simple problem where a single request is made: the request is from Figure 3 where the user `mitchell` would like his mail taken from room 5303 to room 5313. Figure 4 shows the search tree generated by PRODIGY.




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

Figure 4: Search Tree and Solution for single task problem; goal nodes in ovals, executed actions in rectangles.

When the request arrives at the ROGUE module, ROGUE translates it into a PRODIGY state and goal description and then spawns a PRODIGY run. PRODIGY uses its domain knowledge to create a series of actions that will achieve the goal. When the planner has mapped out the plan with enough detail to know its first action, it informs ROGUE, which sends a command to Xavier who starts executing the plan. The first action can be determined when PRODIGY knows that the step will be useful in achieving the goal, and will not be dis-achieved by another action. There are four actions that need to be executed in order to achieve the goal, namely *deliver-item* (node 7), *acquire-item* (node 10), *goto-pickup-loc* (node 13), and *goto-deliver-loc* (node 17). The structure of the goal tree indicates that nodes 7 and 10 should be executed after nodes 13 and 17. Simple reasoning shows that achieving node 17 would be pointless since the action would be immediately undone. As a result, ROGUE starts to execute *(goto-pickup-loc)*. The solution shown in Figure 4 shows the complete ordering of the executed actions.

Each of the actions described in the domain model is mapped to a command sequence suitable for Xavier. These commands are executed in the real-world during the operator application phase of PRODIGY, as described above. These 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. PRODIGY relies on a state description of the world to

```
<GOTO-PICKUP-LOC MITCHELL R-5303>
    SENDING COMMAND (TCAEXPANDGOAL "navigateToG" #(TASK-CONTROL::MAPLOCDATA 567.0d0 2316.5d0))
    ...waiting...
    Action NAVIGATE-TO-GOAL-ACHIEVED finished.

    Verifying Location: R-5303
    LOCATION-VERIFIED R-5303

<ACQUIRE-ITEM R-5303 MITCHELL DELIVERMAIL>
    SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please place Tom Mitchell's mail delivery on my tray.")
    SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please indicate on my keyboard when you are finished.")
Are you finished placing Tom Mitchell's mail delivery on my tray?  (y/i(mpossible)): y
    COMPLETED-ACTION (ACQUIRE-ITEM 1 "Tom Mitchell's mail delivery")

<GOTO-DELIVER-LOC MITCHELL R-5313>
    SENDING COMMAND (TCAEXPANDGOAL "navigateToG" #(TASK-CONTROL::MAPLOCDATA 567.0d0 4115.0d0))
    ...waiting...
    Action NAVIGATE-TO-GOAL-ACHIEVED finished.

    Verifying Location: R-5313
    LOCATION-VERIFIED R-5313

<DELIVER-ITEM R-5313 MITCHELL DELIVERMAIL>
    SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please take Tom Mitchell's mail delivery from my tray.")
    SENDING COMMAND (TCAEXECUTECOMMAND "C_say" "Please indicate on my keyboard when you are finished.")
Are you finished taking Tom Mitchell's mail delivery from my tray?  (y/i(mpossible)): y
    COMPLETED-ACTION (DELIVER-ITEM 1 "Tom Mitchell's mail delivery")
```

Figure 5: Planner/Robot Interaction

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.

There is a large variety of available commands, including those to request or update current location information, to acquire images through the vision camera, and to notice landmarks.

For example, the action *(goto-pickup-loc room)* is mapped to the commands (1) find out the coordinates of the room, and (2) navigate to those coordinates. The command `navigateToG`oal creates a (shortest) path from the current location to the requested location, and then uses probabilistic reasoning to navigate to the requested goal. The model performs reasonably well given incomplete or incorrect metric information about the environment and in the presence of noisy effectors and sensors.

The command `C_say` sends the string to the speech board, and responses ma be used by ROGUE while monitoring execution (described in more detail below).

Figure 5 shows a trace of the interaction between the planner and the robot for the plan shown in Figure 4. Each line marked `SENDING COMMAND` indicates a direct command sent through the TCA interface to one of Xavier's modules.

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 plan
4. Send execution commands to robot, monitoring outcome

We have described above ROGUE's behaviour in the face of a single goal request when no errors occur. The sections below describe how ROGUE handles multiple goal requests, reasoning about prioritizing and interrupting actions, and also how it handles simple plan failures.

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.

4 Additional Behaviours

The capabilities described in the preceding section are sufficient to create and execute a simple plan in an unchanging world. The real world, however, needs

Define: $DK \leftarrow$ domain knowledge
Define: $G \leftarrow$ top-level goal
Define: $PG \leftarrow$ pending goals cache (unsolved top-level goals and their subgoals)

At each PRODIGY interrupt point:
  Let R be the list of pending unprocessed requests
  For each *request* $\in R$, turn *request* to goal:
    - $DK \Leftarrow DK \cup \{$ `(needs-item` *request*`-userid` *request*`-object)`
      `(pickup-loc` *request*`-pickup-loc)`
      `(deliver-loc` *request*`-deliver-loc)` $\}$
    - $G \Leftarrow$ `(and` G `(has-item` *request*`-userid` *request*`-object))`
    - $PG \Leftarrow$ `(and` PG `(has-item` *request*`-userid` *request*`-object))`
    - *request*-completed $\Leftarrow$ nil

Figure 6: Integrating new goal requests into the search tree.

```
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>))
```

Figure 7: Goal selection search control rule

a more flexible system that can monitor its own execution and compensate for problems and failures. In addition, simple single-goal plans such as the one described above are overly simplistic and do not address the needs of the people who will be using these robotic agents. This section describes the extensions we have implemented to the base-line system in an attempt to start addressing real-world issues.

4.1 Interrupts & Multiple Goals

It is very possible that while ROGUE is executing the plan to achieve its first 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 similar requests.

When a new request comes in, ROGUE adds it to PRODIGY's pending goals cache and updates the domain model. Pseudocode for doing the full goal integration is shown in Figure 6. The important points are that (a) the relevant information about the request is added to PRODIGY's domain model, and (b) the new goal is added to the list of pending goals – the goals that must be achieved before the planning is complete.

When PRODIGY reaches the next decision point, it fires any relevant search control rules. Search control rules force the planner to focus its planning effort on selected or preferred goals, as described above. Figure 7 shows ROGUE's goal selection control rule which forces PRODIGY to examine all of its remaining unsolved goals; it is at this point when PRODIGY first starts to reason about the newly added task request. This particular control rule selects those goals with high priority and those goals which can be opportunistically achieved without compromising the main high-priority goal.

The function `(ancestor-is-top-priority-goal)` calculates whether the goal is a subgoal of a high priority goal. ROGUE prioritizes goals according to a simple, modifiable metric. This metric currently involves looking at the user's position in the department and at the type of request: $Priority = PersonRank + TaskRank$. The request also contains deadline information and a "why" slot for additional reasoning to be implemented in the future; this information would allow goal priorities to change with time or situation-dependent features.

The function `(compatible-with-top-priority-goal)` allows ROGUE to identify when different goals have similar features so that it can opportunistically achieve lower priority goals while achieving higher priority ones. For example, if multiple people whose offices are all in the same hallway asked for their mail to be picked up and brought to them, ROGUE would do all the requests in the same episode, rather than only bringing the mail for the most important person. Compatibility is currently defined by physical proximity (*"on the path of"*) with a fixed threshold for being too out of the way, although other features of the domain could (and should) be taken into account.

The control rule feature of PRODIGY permits plans

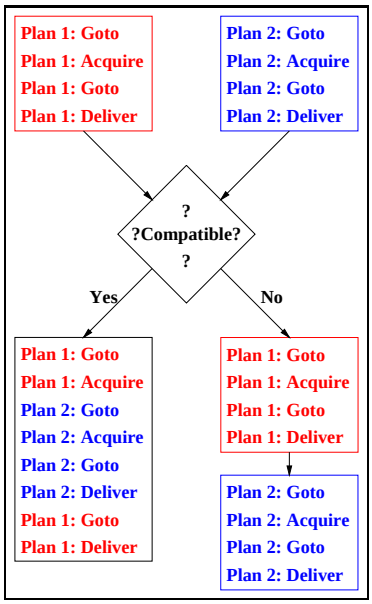


Figure 8: Merging Two Plans

and actions for one goal to be interrupted by another without necessarily affecting the validity of the planning for the interrupted goals. PRODIGY's means-ends search engine simply suspends the planning for the interrupted goal, plans for and achieves the new goal, then returns to planning for the interrupted goal. By using its domain model, PRODIGY is able to identify whether the suspended plan has been invalidated; if so, then it will replan the invalid portion of the plan. PRODIGY's means-ends search engine supports dynamic goal selection and changing objectives by making it easy to suspend and reactivate tasks.

Figure 8 shows how two plans might be merged by PRODIGY's control rules. If the two plans are compatible, ROGUE identifies the order which *most* exploits the similarity between the two plans, and merges the steps accordingly (orderings other than the one shown are possible, and steps may be eliminated if appropriate). If however, the two plans are not compatible, ROGUE suspends execution of the lower priority plan until the higher priority one is complete. When it resumes execution of less important plan, it *does not re-execute unnecessary parts*. If, for example, ROGUE had already acquired the item in question, it would not attempt to re-acquire it; the knowledge of having acquired the object is not forgotten.

The search tree shown in Figure 9 shows how PRODIGY expands the two goals `(has-item mitchell delivermail)` and `(has-item jhm deliverfax)`. The second user (`jhm`) is a more important person, making a more important request. The request arrives via the TCA message interface while Xavier is moving towards room 5303. ROGUE examines the new request and identifies that it is more important than the original (current) goal. However, the current goal not only shares a delivery point with the new goal, but also the physical path of the original goal subsumes that of the new goal. ROGUE decides therefore that the two goals are compatible and that it can achieve the lower priority goal without seriously compromising the new goal. It continues along its path to room 5303, acquires the first object, then moves to room 5311 where it acquires the second object, then completes the delivery of both items to room 5313.

4.2 Monitoring Execution, Detecting Failures & Replanning

Any action that is executed by any agent is not guaranteed to succeed in the real world. 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 any 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.

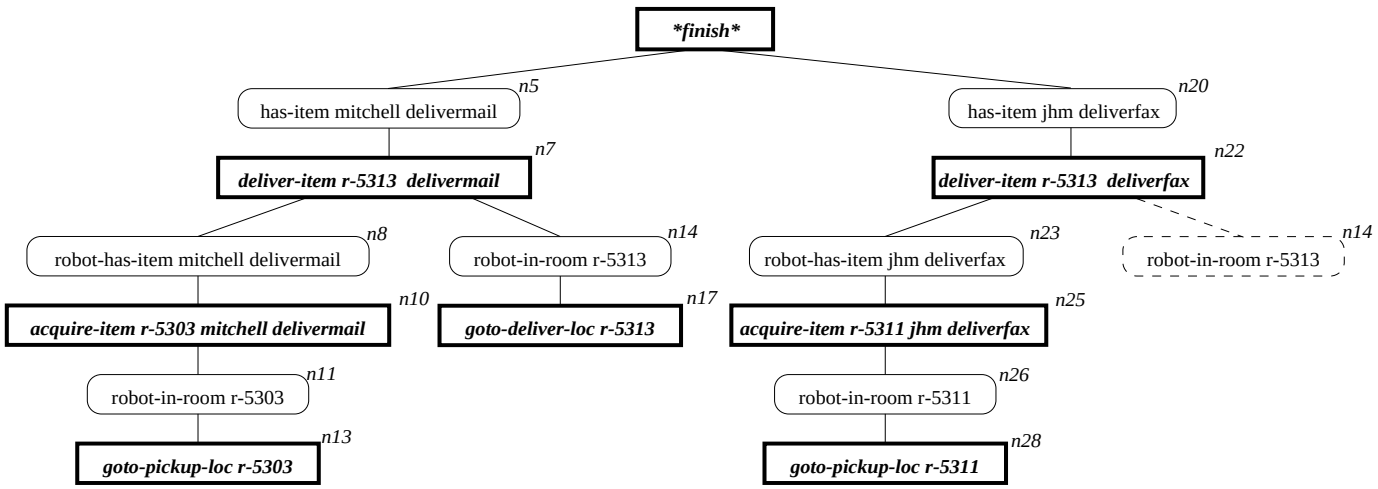
The TCA architecture provides mechanisms for monitoring the progress of actions. ROGUE currently monitors the outcome of the `navigateToG` command. Since the navigate module may get confused and report a success even in a failure situation, ROGUE always verifies the location with a secondary test (vision 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 attempt to find another 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. The process is described in more detail by Stone [15].

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 knowledge 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.

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




```
Solution:  <goto-pickup-loc mitchell r-5303>  -  executed.
           <acquire-item r-5303 mitchell delivermail>  -  executed.
           <goto-pickup-loc jhm r-5311>  -  executed.
           <acquire-item r-5311 jhm deliverfax>  -  executed.
           <goto-deliver-loc mitchell r-5313>  -  executed.
           <deliver-item r-5313 jhm deliverfax>  -  executed.
           <deliver-item r-5313 mitchell delivermail>  -  executed.
```

Figure 9: Search Tree and Solution for two task problem; goal nodes in ovals, executed actions in rectangles.

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.

5 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 [8] 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. It is also not clear how PARETO handles action failure.

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 spacecraft robots [2]. 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.

6 Summary

In this paper we have presented ROGUE, an integrated planning and execution robot architecture. ROGUE has the ability

- 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.

An autonomous agent with all of these features has clear advantages over more limited agents. Although there are a small number of other integrated architectures which support some of these features, none appear to support them all.

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.

This work is the basis for machine learning research with the goal of creating an agent that can reliably perform tasks that it is given. We intend to implement more autonomous detection of action failures and learning techniques to correct those failures. In particular, we would like to learn contingency plans for different situations and when to apply which correction behaviour. We also intend to implement learning behaviour to notice patterns in the environment; for example, how long a particular action takes to execute, when to avoid particular locations (e.g. crowded hallways), and when sensors tend to fail. We would like, for example, to be able to say *"At noon I avoid the lounge"*, or *"My sonars always miss this door... next time I'll use pure dead-reckoning from somewhere close that I know well"*, or even something as apparently simple as *"I can't do that task given what else I have to do."* When complete, ROGUE will learn from real world execution experience to improve its high-level reasoning capabilities.

PRODIGY has been successfully used as a testbed for machine learning research many times (e.g. [10, 18, 16]), and this is the primary reason why we selected it as the deliberative portion of ROGUE. Xavier's TCA architecture supports incremental behaviours and therefore will be a natural mechanism for supporting these learning behaviours.

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, pages 268–272, San Mateo, CA, 1987. 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, Stanford, 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 *Proceedings of AIPS-94*, pages 49–54, Chicago, IL, June 1994.

[5] Erann Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of AAAI-92*, pages 809–815, 1992.

[6] Kristian Hammond, Timothy Converse, and Charles Martin. Integrating planning and acting in a case-based framework. In *Proceedings of AAAI-90*, pages 292–297, San Mateo, CA, 1990. Morgan Kaufmann.

[7] Drew McDermott. Planning and acting. *Cognitive Science*, 2, 1978.

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

[9] Joseph O'Sullivan and Karen Zita Haigh. *Xavier*. Carnegie Mellon University, Pittsburgh, PA, July 1994. Manual, Version 0.2, unpublished internal report.

[10] M. Alicia Pérez. *Learning Search Control Knowledge to Improve Plan Quality*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, July 1995. Available as Technical Report CMU-CS-95-175.

[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. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation*, 10(1), February 1994.

[13] Reid Simmons, Rich Goodwin, Karen Zita Haigh, Sven Koenig, and Joseph O'Sullivan. A modular architecture for office delivery robots. Submission to *Autonomous Agents 1997*, February 1997.

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

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

[16] Manuela M. Veloso. *Planning and Learning by Analogical Reasoning*. Springer Verlag, Berlin, Germany, December 1994. PhD Thesis, also available as Technical Report CMU-CS-92-174, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[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), January 1995.

[18] Xuemei Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proceedings of ML-95*, Tahoe City, CA, 1995.