Adaptive Planning for Distributed Systems using Goal Accomplishment Tracking

Kevin Lee **Nicolas Small** **Graham Mann**

Applied Artificial Intelligence Laboratory,
School of Engineering and Information Technology, Murdoch University,
90 South Street, Murdoch WA 6150, Australia
{Kevin.Lee, N.Small, G.Mann}@murdoch.edu.au

Abstract

Goal accomplishment tracking is the process of monitoring the progress of a task or series of tasks towards completing a goal. Goal accomplishment tracking is used to monitor goal progress in a variety of domains, including workflow processing, teleoperation and industrial manufacturing. Practically, it involves the constant monitoring of task execution, analysis of this data to determine the task progress and notification of interested parties. This information is usually used in a passive way to observe goal progress. However, responding to this information may prevent goal failures. In addition, responding proactively in an opportunistic way can also lead to goals being completed faster. This paper proposes an architecture to support the adaptive planning of tasks for fault tolerance or opportunistic task execution based on goal accomplishment tracking. It argues that dramatically increased performance can be gained by monitoring task execution and altering plans dynamically.

1 Introduction

The aim of many automated and semi-automated systems is to achieve a defined goal. A goal can be defined as a desirable state of the world (Ghallab, Nau & Traverso 2004). Goals are domain-specific and can be physical (Small, Mann & Lee 2013), business-centric (Marks, Mathieu & Zaccaro 2001) or computing related (Deelman, Singh, Su, Blythe, Gil, Kesselman, Mehta, Vahi, Berriman, Good, Laity, Jacob & Katz 2005). Completing a goal amounts to choosing and then performing those actions that are required to change the state of the world from the current state to the desirable goal state. A complex goal is completed through the navigation of a plan consisting of a tree of interdependent tasks, with each parent task relying on the fulfilment of all its child tasks.

A completed goal might lead to a physical action, a log entry, a human notification or a trigger for another process. To achieve a goal it is normal that a series of tasks must be accomplished, often in a workflow (Deelman et al. 2005, Ghallab, Nau & Traverso 1995). Knowing the current state of the goal progress is important to determine if everything is going smoothly or if there is a problem.

Goal accomplishment tracking is used to monitor a series of tasks that are intended to meet a goal. Tasks are monitored, intermediate and final results are analysed and the progress of the goal is calculated. The outcome of this process is normally a simple completion notification. Monitoring of sub-goals is useful in a number of fields, such as monitoring tasks in scientific workflow execution (Deelman et al. 2005), monitoring of teams in business team management (Marks et al. 2001) and monitoring robotic exploration (Small et al. 2013).

Goal accomplishment tracking can be applied to any area that uses a series of tasks to attempt to meet a high-level goal. For a simple goal with a small number of tasks, Goal accomplishment tracking can be used to give a simple indication of progress, by e.g. parsing log files. For a complex workflow based goal, it can be used to reveal complex task relationships.

Although generally used for passive informational reasons, goal accomplishment tracking has the potential to trigger a response in the event of a goal completion being at risk. It can also trigger responses in the event of a potential opportunity to improve the goal completion time. These events are likely when there are incorrect assumptions in the task descriptions or unforeseen environmental changes.

Reacting to problems and opportunities involves changing what is currently happening in terms of the task plan. This could mean a number of different actions (Lee, Sakellariou, Paton & Fernandes 2007), including creating new tasks, removing tasks, choosing alternate implementations or reallocating resources - all of which could be called adaptive or autonomic planning (Ghallab et al. 2004). Adaptive planning in this scenario broadly takes into account the current environmental conditions together with the current plan and creates a new plan. This new plan usually takes into account domain-specific constraints such as resource performance and cost (Lee, Paton, Sakellariou & Fernandes 2011).

This paper proposes the use of goal accomplishment tracking as a trigger for adaptive planning in automated systems. In the event of the discovery of a problem or opportunity, an adaptive planner is triggered to analyse the current progress of task execution and calculate a new plan. This has potential to not only react to potential goal failures, but also taking advantage of opportunities to dramatically decrease the time needed to achieve a goal.

The remainder of this paper is structured as follows. Section 2 presents a background on goal-based systems. Section 3 introduces goal tracking. Section 4 discusses the role of planning and introduces

adaptive planning. Section 5 introduces a proposed architecture for supporting adaptive planning as the result of goal accomplishment tracking analysis. Section 6 presents a case-study based evaluation that gives contexts to the proposed approach. Finally Section 7 presents some conclusions and discusses future work.

2 Background

The idea of goal-seeking as a principle to guide intelligent behaviour emerged early in the history of artificial intelligence, and with it the notion of goal satisfaction. Only a few years after the birth of the science, notions of analysing problems by means-end analysis, or recursive simplification of compounds to constituents elementary enough to be solved by trivial means were being formulated in the hope of discovering general-purpose methods for problem solving (McCarthy 1963, Newell & Simon 1961). Once the language of plans, goals, world-states and operators was established, ideas about how to automatically generate plans - ordered sequences of operators capable of systematically transforming observed world-states into desireable world-states began to appear (Winston 1984). In this context, notions of tracking goal accomplishments are normally expressed in rather general perceptual terms, with the details only discussed at the level of particular implementations, if at all. A separation between planning and monitoring in robots was proposed by Munson (Munson 1971), and later expanded in (Noreils & Chatila 1995), but in these works, monitoring occurred in the execution loop of the robot, working at a level too low to satisfy any need for distributed, multi-agent coordination. Both of these monitoring systems also focused on error detection rather than success detection: unless an error was detected, the robot was assumed to have succeeded in its task. The monitoring proposed here is geared toward detecting success first and foremost, only initialising error detection and potential replanning if success is not achieved.

SRI's Shakey represented an important embodiment of this kind of deliberative planning - also called the sense-plan-act paradigm - in a physical robot (Nilsson 1984) capable of carrying out plans in controlled static environment of wooden boxes. It showcased Stanford's classical automated planning system, STRIPS (Fikes & Nilsson 1972), which could break down task-oriented command expressions into ordered sequences of actions selected from the robot's repertiore, which it would then execute. Although Shakey was slow and could not deal with dynamic, complex environments, later developments of the sense-plan-act concept began to tackle these problems. Additional layers would be introduced in the planning part, any of which might draw on the world model. Some layers were intended to generate future plans - hierarchies of goals. These plans would be decomposed into simpler commands before being sent to lower levels for execution. Lower layers would then further decompose the commands into actionable tasks which would ultimately be passed on to actuators at the lowest level. The sensing layer is designed to keep the internal representation of the world up to date. This was discovered to be difficult to do in any realistic dynamic enviroment, and intervening planning layers were later added to suggest changes to plans based on recent changes to the world model.

It is usually convenient to represent goals in the same form as observed states of the world as provided by sensors. Checking the accomplishment of a goal might then be as simple as comparing it with a current set of world-state representations. For example, if a robot is equipped with a sensor which returns its position in two-dimensional space, a locational goal might be specified as 'robot_at(250, 300)'. This can be directly compared to the output of the sensor, which might return 'robot_at(240, 300)'. A more sophisticated arrangement would involve abstracting the goal to 'robot_at(waypoint)' and providing additional knowledge defining the waypoint in terms of a number of sensory tests and satisfaction condition with respect to those tests. In this case, the waypoint is associated with a set of GPS coordinates of an area within which the target is found, a specific barcode known to be present at the target an image pattern to be matched against a known landmark through the use of an on-board camera. The satisfaction condition is that the current GPS coordinates must be within bounds, and that a match on either of the other sensory indicators would be sufficient. Such straightforward arrangements would not always suffice, because the relationship between sensory data and actual world states is not always this simple. Not only does the sensitivity, range and signal-to-noise ratio characteristics of a given sensor affect the interpretation of its signals, but the satisfaction or failure of some goals might involve a subtle alteration in sensed properties, possibly including necessary state progressions or alternatives. Some of the literature on robot perception deals with the control of uncertainty introduced by these complications (Thrun, Beetz, Bennewitz, Burgard, Cremers, Dellaert, Fox, Haehnel, Rosenberg, Roy et al. 2000, Minguez, Lamiraux & Montesano 2005).

Futhermore, not all goals are the same, but may have different natures based on their objective and relation to processing. In this paper we distinguish three types of goals: i) achieve goals, which are simple expressions denoting a desired world state to be reached; ii) maintenance goals, which need to be protected (i.e. their accomplishment tests must not be allowed to fail); iii) and opportunistic goals those associated with a watch for particular events or world-states, the presence of which is considered favourable.

All sense-plan-act systems depend on the system's ability to keep track of the state of the world. In mathematical abstractions, data from a simulation, model or from the real states of other software elements, could be fed in at each planning cycle. In physical robots, input would be made via a perceptual system informed by electronic sensors, and updated at every sensor sweep. Early models assumed the availability of complete and accurate information about the world (Schwartz, Sharir & Hopcroft 1987, Canny 1988, Latombe 1996). In systems that realistically modelled the real world, it quickly became clear that building and maintaining world models of sufficient breadth, detail and accuracy, and doing it quickly enough to be useful, were very difficult problems. In distributed systems, the problem is further complicated by the need to reconcile the possibly different world-models of multiple agents or robots, so that cooperation can be achieved. Difficulties of this kind eventually led some robotics researchers to move away from the sense-plan-act paradigm, first to the idea of entirely model-free, reactive systems, such as behaviour-based control (Brooks 1986, Mahadevan & Connell 1992) then to control based on simpler, more tractable mathematical formalisms, such as probability theory (Cassandra, Kaelbling & Kurien

1996, Thrun et al. 2000), and later to hybrid systems which attempted to combine the strengths of low-level reactive control with those of a command layer that could reason about plans (Gat et al. 1998, Montemerlo, Becker, Bhat, Dahlkamp, Dolgov, Ettinger, Haehnel, Hilden, Hoffmann, Huhnke et al. 2008). These typically used minimal interlayer communication pipelines to try to reconcile the two dissimilar approaches.

But there are ways of improving the efficiency and reducing the burden of world-state modelling without abandoning the traditional deliberative paradigm. An update of the Shakey system, which overcome its well-known failure in dynamic environments by interleaving sensory layers and planning layers and by using a superior logic formalism that avoids the frame problem was proposed (Shanahan 2000). The STRIPS planning language has also been improved (Bonet & Geffner 1999), but the defacto standard planning language is now perhaps the Planning Definition Domain Language (PDDL-3 or a variant of it) (Gerevini & Long 2005) by virtue of its use in the International Planning Competitions (Gerevini, Haslum, Long, Saetti & Dimopoulos 2009). Given sufficient computational resources, the world modelling problem can be delegated to specialised modules organised in layers. The best of these designs is probably 4D/RCS (Schlenoff, Albus, Messina, Barbera, Madhavan & Balakirsky 2006). The 4-Dimensional Reactive Control System architecture is a six-deep hierarchy of goal-directed, sensory intelligent control processes. Based on the idea that several different representations of the world can be be combined in synergistic ways, the system uses procedural knowledge represented as production rules, while declarative knowledge is represented in classes, frames and semantic nets. It also includes signals, images, and maps in its knowledge-base. Representation-crossing processes maintain a close coupling between iconic and symbolic data structures in real-time. This system has been the basis of a number of projects such as the US National Institute of Standards reference architecture for autonomous vehicles.

Erann Gat argued that the difficulty of timely and accurate world modelling is due to improper control of the time scales and levels of accuracy over which model assumptions are presumed to hold true. By leaving rapidly-changing states of affairs out of plans, modeling world states at non-specific abstract levels for overall guidance, and then relegating the detailed control of actions to sensor-based action programs, these difficulties could be greatly reduced (Gat 1993). His control architecture based on these ideas, ATLANTIS, proved successful in simulations and a number of real-world robots (Gat 1991). Another idea is based on an analysis of the complexity of computing a control strategy for sensory-motor tasks under various levels of sensor and movement uncertainty. According to Erdmann (Erdmann 1995), in the case of perfect sensors, the problem is relatively easy (polynomial time on the number of object constraints) easy, even under uncertain action control. This led to the concept of building a backchaining planner based on the ideal of perfect sensors, and designing these to be specific to a given task. Since actual construction of perfectly informative sensors is impossible, the question becomes one of how tolerant the planner is of real, imperfect sensors that approximate the ideal. He concluded that such sensors should recognise actions that fall within 'cones of progress' toward goal end-points, rather than states themselves. It is not clear that this idea ever went any further, possibly because few researchers in this area are active in sensor design.

When tasks are distributed in multi-agent or multi-robot systems, keeping track of goals becomes more complicated, even while more computional power becomes available for the purpose. Our approach is to quickly and efficiently compile logical expressions that are highly constrained by available sensors and easy-to-specify, abstract knowledge of their relationship to important world-states. These goal accomplishment sensing opportunities are tasks that can be offered to multiple agents or robots in a distributed system. One or more of these may take on a single or multiple goal accomplishment detections over the duration of plan execution.

3 Goal Accomplishment Tracking

All sufficiently complex automated and semi-automated (those with human intervention) activities require the use of a plan which describes what is to be done. Different domains use different types of plans, but they all fundamentally consist of a set of interdependent sequential sub-goals. They all have in common the aim of achieving a final result or *Goal*.

In robotics, a plan consists of a set of observabl world-states which the robot can accomplish by means of navigation, manipulation and perception, to e.g. perform a maintenance activity (Small et al. 2013). In scientific computing, a plan consists of a set of executable tasks connected together in a workflow to be executed on large-scale computational resources (Deelman et al. 2005).

A useful way of abstracting these activities generically is to represent them as a goal tree. Figure 1 illustrates this view of an activity in terms of goals and sub-goals. There are two types of goal-trees showed here: a linear one and an acyclic directed graph or workflow. Before each goal state is achieved, each goat state it is dependent on must have successfully been achieved.

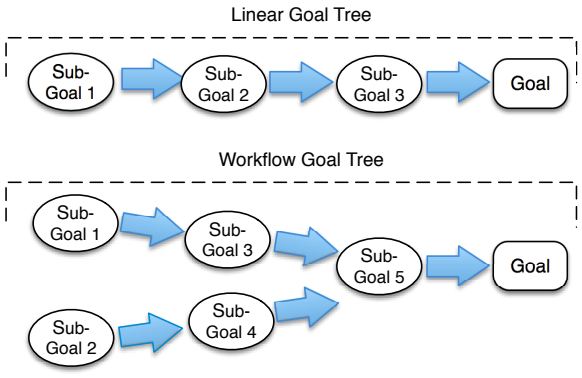

Figure 1: Plans, Tasks and Goals

A goal-tree has goals, represented as nodes of the goal tree, of the form$< n, g^s, t^i >$, where n is the id of the goal, g^s is a description of the goal target state and t is a set of i of goal accomplishment tests. For a goal to be considered accomplished, some or all i of t tests need to be successful. At this stage the goal is accomplished and the process for completing the next goal can begin.

There are broadly, three types of goals (Small et al. 2013).

- **Achieve goals** (Dastani, Riemsdijk & Meyer 2006) are simple expressions denoting a desired world state to be reached.
- **Maintenance goals** (Dastani et al. 2006) are goals that need to be protected (i.e. their accomplishment tests must not be allowed to fail). Maintenance goals require extra monitoring after they have been initially accomplished, placing an extra burden on the monitoring system.
- **Opportunistic goals** are goals associated with a watch for particular events or world-states, the presence of which is considered favourable. Opportunistic goals mirror maintain goals, in that rather than demand checks for threats to goals, they encourage checks for contingencies favourable to goal accomplishment.

To track the progress of a goal-tree with respect to the overall goal, the progress of each goal must be monitored. The overall progress of the plan can be called *goal accomplishment tracking* as it refers to the plans progress towards the overall goal - this is distinct from goal tracking which is concerned with the goals themselves. This means testing at intervals all of the tests in t to determine their progress towards the accomplished state. When some or all of the tests indicate the goal is accomplished, then it is complete.

To determine the goal accomplishment progress whilst the plan is being enacted, the status of each goal must be calculated. This is domain-specific and generally involves the keeping of a goal completion log which is updated each time a task completes. This is quite a coarse-grained data source for determining the overall progress of the goal, ignoring any progress during execution. More fine-grained detail of the progress of the goals can only be achieved by measuring the course of the execution directly.

Monitoring goal progress during task execution can be done in two different ways, i) by using programmed checkpoints as part of the mechanism used to complete the goal itself or ii) by observing the world. Programmed checkpoints depend on data from the entity that is attempting to support the completion of the goal. The entity, such as a physical robot or a software program, needs to report back at regular intervals on the progress towards the goal.

The progress of a particular goal can also be determined not just in an active way, but also in a passive way through observation, not necessarily from the entities attempting to achieve the goal. Depending on the task, a variety of sensors can be used to measure progress. For example, for a robot navigation task by Euclidean distance between pairs of way points, a GPS sensor on the robot can be used to provide an estimate of the task progress. Likewise, the progress of a data processing task can be determined by the memory or storage usage. The most effective way of determining the progress of tasks towards the overall goal may be a combination of these two approaches.

This information provides a rich source of data to search for patterns in the activities execution. The most likely pattern - other than a normal execution - is that of a problem or fault with the activity. This is often due to unforeseen environmental conditions in the particular domain or incorrect assumptions in the goal-tree itself. In robotic navigation, a GPS black spot could cause a navigation task failure. In scientific workflow execution a task could be delayed due to resource contention from other workflows. All of these issues could be seen as a risk to the timeliness of individual tasks and the overall goal.

In addition to looking for potential problems during the execution of a goal-tree, if the progress of goals is being actively monitored then there is the possibility of looking for opportunities. Opportunities might arise during the execution of a goal-tree to improve the performance or accuracy of the tasks, and their implementation by altering it. Such an occurrence might be due to the plan being too conservative, activities already being completed by a third party or implementations completing more than one goal simultaneously. Taking advantage of opportunities as they arise might reduce the time it takes to reach the goal or make the cost less than expected.

Goal accomplishment tracking alone cannot deal with potential problems or opportunities other than notifying a supervisor or supervisory system. Any supervisory system must take into account all plan and environmental factors and decide what to do. This paper takes the view that in this context, a goal tracking system would create triggers upon the discovery of plan problems or opportunities. These triggers would result in continued execution, halting of the plan, human intervention or adaptive planning.

4 Planning and Adaptive Planning for Goals Achievement

Planning is the activity of functionally decomposing goals and mapping them into a series of concrete tasks to produce a concrete plan as illustrated in Figure 2. This can lead to a plan with physical tasks, human interaction, software execution, or a combination of these. For example, a robotic navigation exercise with a location-based goal will be planned to have a series of tasks to get the robot to that location, such as following a particular path or turning the wheels at a specific speed. Planning a scientific computing activity such as building maps of the sky (Deelman et al. 2005) will involve a plan containing tasks for moving data around, normalising data sources and executing many programs on remote machines.

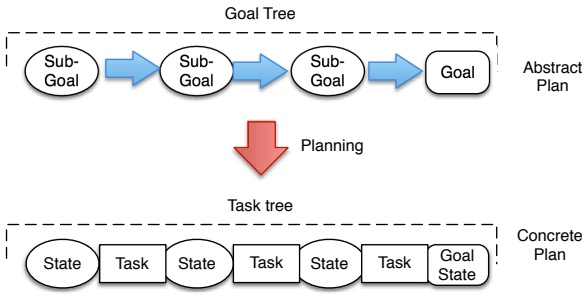

Figure 2: Using Planning to produce a task plan from a goal tree

These activities share the same aim; at the end of this planning process, the plan should be concrete and usable; with the result of its execution leading to the successful accomplishment of the goal. Planning can be either a manual or automatic process. In a manual planning process a person must define the task to be completed at each stage of the plan. The intelligence of task implementation (the robot, human or software), determines how precisely the tasks need to be described. Manual planning is common in areas involving people or for those goals involving plans

that rely on replay, such as with repeating tasks of industrial robots in controlled environments.

In the process of planning, an abstract goal tree with nodes of type $< n, g^s, t^i >$, needs to be converted to an actionable and concrete plan. For the purposes of this paper, a plan may be described as having nodes of the form $< t, s^{start}, s^{end}, a^{function}, a^{human}, a^{instruction} >$ where t is the id of the task, s^{start} is a description of the start state and s^{end} is a description of the desired end state. The remaining items are actions, $a^{function}$, a^{human}, $a^{instruction}$ which are lists of software functions, human actions or physical instructions respectively.

An edge in a plan is of the form $< t_1, s_1^{end}, t_2, s_2^{start} >$ and expresses a dependency between the target goal state s_1^{end} of a task t_1 and the initial state s_2^{start} of a task t_2. A task is a logical path from one goal state g^s1 to another goal state g^s2. The successful completion of the task would lead the system to progress from g^s1 to g^s2. This is illustrated in Figure 2 as the transitions in the goal tree between sub-goals being planned as tasks in between states of the system.

Automated Planning is the activity of producing a plan in an automated way with no human interaction. To perform automated planning, a piece of software, called a planner, must take into account the final goal, the goal tree, the environmental conditions and the availability of resources. The planner will calculate the most efficient plan possible, and may take into account secondary issues such as resource contention or cost.

An extension of automated planning is that of adaptive planning. Adaptive planning is the task of re-planning whilst the plan is in the process of executing. So, unlike design-time planning, an adaptive planner will also take into account the current state of the goals and sub-goals. During execution of the plan, environmental changes cause the plan which was potentially optimal to become sub-optimal, thus making it desirable to create a new plan. When re-planning, it is important to take into account the work that has already been done, so as not to repeat effort. But note that a system that efficiently handles opportunistic goals should not be troubled much by this, since it would quickly detect existing accomplishment and move onto the next work that was still needed.

Adaptive planning in response to task execution performance takes intro account the environmental conditions that can affect the task. In the area of workflow execution (Lee, Paton, Sakellariou, Deelman, Fernandes & Mehta 2009, Lee et al. 2011) adaptive planning is useful as there is ongoing and dynamic contention for cluster-based computing resources. Any decision that is made statically during a planning or automated planning process can be revised during adaptive planning (Lee et al. 2007). Adaptations can be initiated and deployed for many reasons, including:

- Changes in the environment e.g. availability of resources.
- Changes to the goal or sub-goals e.g. if an unexpected obstacle appears to the goal.
- Changes due to the availability of tasks e.g. to available robots or people

Changes to the environment, such as changes in weather or road surface in robotics, or availability of processing resources in workflow execution, may prompt adaptive planning. Dynamic changes to goals or sub-goals are possible during runtime, forcing adaptive planning to meet the new goals. Finally, changes to the availability of agents to support goal accomplishment will force adaptive planning so that the available agents are utilised. Adaptations could be initiated for different reasons, including reactive (based on monitoring events) and proactive (based on prediction of future performance). In the approach and system proposed in this paper, adaptations are initialised for opportunistic performance and fault tolerance reasons.

Adaptive planning in support of goal accomplishment, shares techniques in common with other adaptive and autonomic computing areas (Kephart & Chess 2003). The main characteristic of adaptive and autonomic computing work is the use of a feedback loop to structure the implementation of the adaptive support software. This feedback loop, which has the primary purpose of collecting data and using it to aid in the adaptive planning process is implemented differently in different adaptive systems, but fundamentally occurs in four phases. The *Monitoring* stage collects information about the environment, agents and progress towards the goal. The *Analysis* phase uses the collected information to determine if there are opportunities or problems which might benefit from adaptive planning. The *Planning* phase will explore the range of possible alternative plans to determine which plan should be used. Finally, the *Execution* phase deploys the new plan.

In the context of this paper, the monitoring phase involves the collection of a variety of information during runtime. The information that must be collected involves the current progress towards the goal - this information can be collected actively from participating agents, but also passively by using sensors observing the environment. Information about the environmental conditions is also collected and flagged if the environment has changed. The state of the agents performing tasks are also monitored. The analysis phase looks for goals potentially heading for failure as well as potential opportunities as with Erdmann's "cones of progress" (Erdmann 1995). In these events, Planning is undertaken adaptively based on the available up-to-date information. Execution involves the replacement of tasks aiming to lead to completed goals.

5 A Proposed Architecture for Goal Accomplishment Tracking based Adaptive Planning

5.1 Overview

For activities that can be abstracted as a series of goals in a goal-tree, goal accomplishment tracking introduces the possibility of revealing the ongoing progress towards the goal. This fine-grained information allows the detection of potential problems and opportunities that can effect progress towards the final goal. Acting on these triggers could lead to faults being detected and rectified, and opportunities being acted on, reducing the time to goal. This section introduces a proposed architecture for retro-fitting adaptive planning to existing systems using goal accomplishment tracking to produce these triggers. Section 5.2 introduces the proposed architecture. Section 5.3 discusses the range of potential adaptive planning types. Section 5.4 provides a summary.

5.2 The Proposed Architecture

Figure 3 illustrates a layered view of the proposed approach of adaptive planning with goal accomplishment tracking. The lowest layer is the execution layer, which includes the world in which the system operates, the system agent information and sensors that provide observational data. This is passed up to a supervision layer which includes goal accomplishment tracking and sensor management. The management of the plan is handled in the layer above. The plan manager can only manage a current plan, so to support adaptive planning another layer is needed above plan management - as illustrated in Figure 3. The progress tracker receives updates from the execution layer and uses the trigger mechanism to call the adaptive planner.

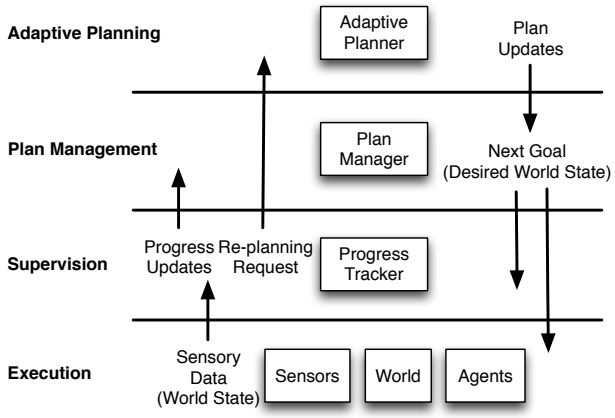

Figure 3: Layered approach to adaptive planning with goal accomplishment tracking

Figure 4 illustrates a proposed component-based architecture for adaptive planning using goal accomplishment tracking. The figure illustrates that the *Target System* is being managed using a plan and data is being collected from sensors for the purpose of goal accomplishment tracking.

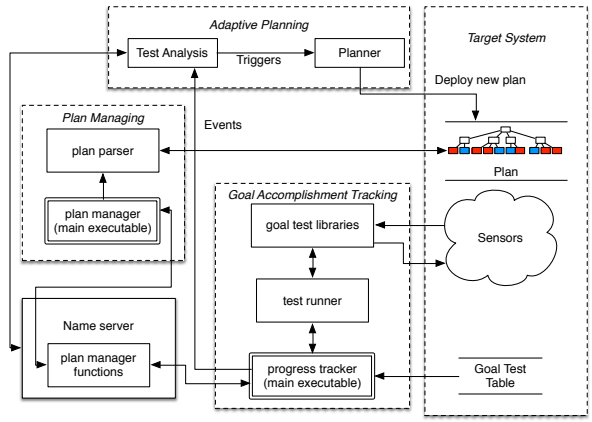

Figure 4: Goal Accomplishment Tracking and Adaptive Planning

The *Goal Accomplishment Tracking* components track the progress of goals by using a set of goal test libraries. The use of a library containing multiple tests allows fine grained monitoring of goal progress. This information is passed to the *Plan Managing* components, which correlates it against the deployed plan to determine the current state of the plan - and therefore the goal accomplishment progress.

Adaptive Planning consists of two components that support the analysis of the current progress of the goal and adaptive planning. The test analysis component and plan manager allow the adaptive planner to react to problems and opportunities triggered by goal accomplishment tracking. The test analysis component creates triggers based on goal accomplishment tracking and feeds these to an adaptive planner. The adaptive planner takes into account environmental conditions, the current state of the goal-tree and produces a new plan which will be used in further execution.

The *Test Analysis* component is illustrated in more detail in Figure 5. It illustrates the processes involved in the analysis phase of adaptive planning. In normal operation, the goal accomplishment tracking sub-system correlates progress updates from sensors with the goal table to determine the goal accomplishment status. When called, adaptive planning can use this same data-set and dynamically re-plan at runtime.

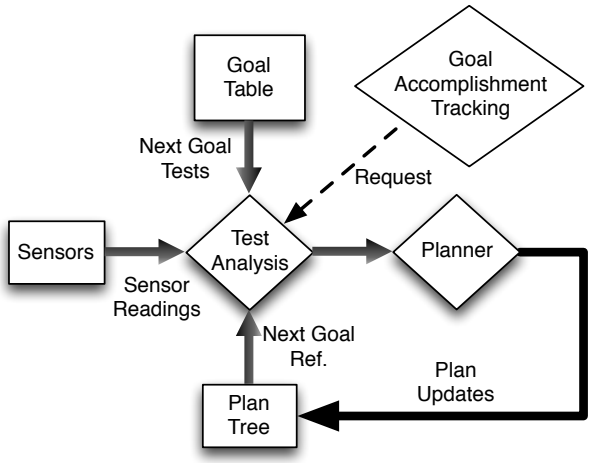

Figure 5: Analysis Phase of Adaptive Planning

5.3 Adaptive Planning

There are a range of potential adaptations, which can result from the adaptive planning process in the proposed architecture. These are broadly in two groups, discussed in the following sections.

5.3.1 Goal to Task Mapping Adaptations

In mapping adaptations the mapping from the goal tree to the concrete tasks changes depending on the environment. Inspiration can be taken from previous work in adaptive workflow processing (Lee et al. 2007) which looks at adapting the execution of abstract scientific workflows by using mapping and scheduling adaptations. Possible mapping adaptations are:

- **Change goal to task mapping.** This changes the current goal/concrete task mapping to use different targets for t and a in the concrete plan. In other words, this replaces the task(s) used to implement the goal with a different task s' or one with different actions a'.
- **Reduce the number of tasks for the goal (task-reducing).** This is where a task t is mapped to fewer tasks $t'_1, \ldots, t'_1$ and actions $a'_1, \ldots, a'_1$ than before. This may be appropriate

when, e.g., overheads per task instance are high and the performance of one task have increased to the point that the task need no longer be split across as many task instances.

- **Increase the number of tasks for a goal (task-splitting).** In this case, the number of tasks $t'_1, \ldots, t'_n$ onto which a goal g is mapped is increased. This may be useful, e.g., if more resources are now available and overheads per service instance are low.
- **Remove a task** A node $< t, s^{start}, s^{end}, a^{function}, a^{human}, a^{instruction} >$ can be removed if its removal does not compromise the correctness of the workflow. This may occur, for instance, if the goal is already complete, negating the need for the task.
- **Change the agent performing the task.** This involves changing the agent who is performing the task t or the actions a. Changing the agent may be appropriate in the event of one agent being more appropriate to complete a task or a set of actions, or due to thr Incapacitation of the active agent.

5.3.2 Task Scheduling Adaptations

There are also adaptations that manipulate the scheduling of the tasks. Adaptive scheduling alters the scheduling policy in response to changes in the environment. Possible task scheduling adaptations are as follows:

- **Increase the level of parallelism of a Task.** For a Task t that is parallelisable, a potentially useful adaptation is to increase its parallelism level by scheduling more than one task instance. This may be possible depending on the availability of resources and the cost to increase the parallelism.
- **Decrease the level of parallelism of a Task.** It may otherwise be appropriate to decrease the parallelism level of a task t by reducing the number of task instances that are scheduled. This may be appropriate if, e.g., the resources for the goal have to be reduced (e.g., because they must be reallocated to same other task with higher priority).
- **Restart Task.** This may be appropriate if changes have been made to the configuration of a task t at a location l, and the task needs to be restarted to activate the changes.
- **Pause Task.** This involves temporarily stopping the execution of a task s or actions a. The execution node can then be used to execute other service(s) with the released resources. Pausing services may be useful to control the overall performance of workflow execution in order to, e.g., enable other services to catch up.
- **Move tasks between agents.** For tasks that can be executed by different agents, it may be useful to move the task between agents. This may be desirable if agents fail, become in demand or become more expensive, or if it is beneficial to optimise the overall task allocation.

This range of potential adaptations provides a rich toolkit for acting upon triggers received from the goal-accomplishment tracking system. Both problems and opportunities can be reacted to through the use of these adaptations.

5.4 Summary

This section has presented an approach to adaptive planning for use with goal accomplishment tracking. It has proposed an architecture for monitoring tasks to determine goal progress and create triggers on potential problems and opportunites. It has also described the range of adaptations possible with this approach.

6 Case study: Assisted Teleoperation

6.1 Overview

This section presents an adaptive planning feasibility analysis of a system that uses goal accomplishment tracking to achieve its goals. The chosen domain is assisted teleoperation for the automatic maintenance of physical equipment by robots. Automatic maintenance of physical equipment requires a mobile robot to periodically visit a number of worksites. The robot may perform tasks such as photography, gathering sensor data on environmental conditions, physically probe the integrity of surfaces, joints or attachments, remove panels and/or to change out faulty components (Mann 2008). To perform this task, a robot needs to be guided around multiple worksites, aligning itself close to each one in turn so as to be able to deal with important objects. Section 6.2 describes the architecture of he assisted teleoperation system and the modifications necessary to enable opportunistic and reactive adaptive planning. Section 6.3 describes the opportunistic and reactive adaptations possible with this scenario. Section 6.4 presents a summary.

6.2 System Description

To investigate the suitability of adaptive planning for this case study, a working plan-based system was analysed. The architecture of the system is illustrated in Figure 6. It is implemented in Python, with communication provided by Pyro middleware.

The system consists of three physical devices communicating over a wireless network using a router mounted in the mobile robot. These devices are i) a 7-inch Android tablet with built-in joysticks and buttons, ii) a laptop, and iii) a mobile field robot. On the software side, communications between components are organised via Python Remote Objects (Pyro). Each component publishes its services with Pyro, enabling inter-component calls. There is also a video feed from the robot, which is broadcast using UDP.

The mobile field robot, a heavily modified Coroware Corobot, is built on top of a Lynxmotion base with four driven wheels, with an Intel D2700MUD Mini-ITX single-board computer running Debian Linux. It is equipped with a wide angle camera, a modified 5 DoF manipulator arm and is equipped with a variety of sensors such as GPS and IR rangefinders.

The human interface is centred around the Android tablet, which displays plan progress, video, and sensor readings from the robot. The operator can use the joysticks and the touch screen to command the

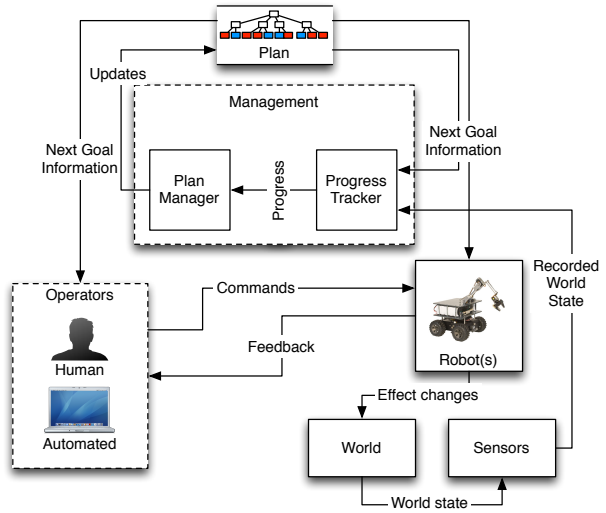

Figure 6: Case Study of Adaptive Planning in Assisted Teleoperation

robot. The automated controller is mainly run on the laptop, where it has access to the most processing power. Some lightweight components are located where they are most needed, cutting down latency issues. This is the case for the inverse kinematics module which is mostly called by and situated with the teleoperation interface.

In this case study, one or more robots are tasked with completing a series of maintainance tasks with the supervision of a human operator. An example plan for an assisted teleoperation-based maintainance task is illustrated in Figure 7. This particular plan is for the robotic replacement of a circuit board using a motorised screwdriver. The main goal of changing the board is split into four sub-goals; checking the needed parts, a navigation exercise, removing the panel and replacing it. Each sub-goal has a number of corresponding tasks that have been initially planned - which are assigned as either robot or human tasks.

In this implementation, plan progress is tracked using two distinct components, the progress tracker, and the plan manager. These components are both run on the laptop. The progress tracker is a Python implementation of the goal accomplishment tracking architecture (Small et al. 2013), using the SURF OpenCV library for video feed testing, simple algorithmic tests for all other text-returning sensors, and a CSV file to store the goal accomplishment tests. The plan manager is a simple Python program charged with updating the plan XML structure. Adaptive planning is accomplished with Python components also on the laptop.

6.3 Adaptive Planning for Assisted Teleoperation

The proposed architecture allows the currently allocated tasks and actions to be altered at runtime to achieve the teleoperation goal. As described in Section 6.2, there are a variety of sensors that can be used to provide information suitable to support the adaptation process. Figure 4 illustrates that the goal tests in the goal test library provide data to the *Test Analysis* component of the proposed architecture. The *Test Analysis* component looks for potential problems and opportunities in the current progress of the goal completion. In the event of a potential problem or opportunity being detected, a trigger is sent to force adaptive planning. There are a variety of possible adaptations that can be performed in assisted teleoperation, these are described in two categories here.

6.3.1 Goal to Task Mapping Adaptations for Assisted Teleoperation

- Change the agent type who is performing a task. This might mean assigning a robot to perform a task which is currently assigned to a different robot; or this might mean having a human supervise a particular robotic activity which was previously completed with no oversight.
- Reduce the number of tasks assigned to a goal. This is most likely to be necessary when tasks become unnecessary to complete a goal - for instance, when a direct path is used instead of the previous planned longer path.
- Increase the number of tasks assigned to a goal. This is most likely to be necessary when there is an unexpected obstacle in the path of the teleoperated robot. The adaptation for this would involve the addition of either an automated task or a human to intervene in the task.
- Removal of a task or group of tasks. If a goal is deemed unnecessary, for instance if it has already been completed by another entity, then a task or group of tasks can be removed.
- Change the agent who is performing the task. This involves the changing of the agent to an agent of the same type. For example, this could involve the replacing of a robot with another who is performing better.

6.3.2 Task-Based Scheduling Adaptations for Assisted Teleoperation

- Increase the level of parallelism of a task. In this case study, if a task is not being performed sufficiently well, more robots can be assigned to the task.
- Decrease the level of parallelism of a Task. Likewise, in this case study, if a task performance is exceeding expectations, the number of robots currently being assigned to the task can be reduced, thus freeing up these resources for elsewhere.
- Restart task. In the event of a task being attempted with errors, the task can be restarted. For example, if a circuit board is incorrectly placed in a slot.
- Pause task. This is most likely in this case study when some tasks require other tasks to be completed first and this hasn't been anticipated. In this case the tasks and/or the agent performing the task can be paused.

6.4 Summary

This section has presented a case study of assisted teleoperation as a demonstration of the possible effectiveness of adaptive planning with goal accomplishment tracking. It has illustrated that adaptive planning can be retrofitted to architectures that are goal plan-based and that this can be useful in looking for and reacting to problems and opportunities.

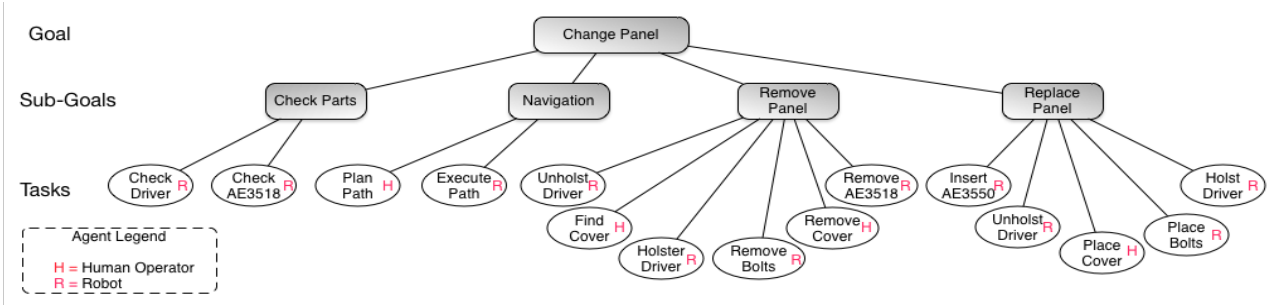

Figure 7: Example Assisted Teleoperation Maintainance Plan

7 Conclusions and Future Work

This paper has argued that goal accomplishment tracking combined with Adaptive Planning is a powerful and flexible technique for supporting goal plan-based automated and semi-automated distributed systems. It has proposed an architecture for monitoring the goal accomplishment progress of systems, looking for potential problems or opportunities and adaptively re-planning if necessary. It demonstrated this approach through a case study of assisted teleoperation. Future work will expand the evaluation with different case studies, more sensor sources and more complex adaptivity decision reasoning. It will also look into proactive adaptations to prevent any potential issues before they arise.

References

Bonet, B. & Geffner, H. (1999), 'Functional strips: a more general language for planning and problem solving'.

Brooks, R. A. (1986), 'A robust layered control system for a mobile robot', *Robotics and Automation, IEEE Journal of* **2**(1), 14–23.

Canny, J. (1988), *The complexity of robot motion planning*, MIT press.

Cassandra, A. R., Kaelbling, L. P. & Kurien, J. A. (1996), Acting under uncertainty: Discrete bayesian models for mobile-robot navigation, *in* 'Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on', Vol. 2, IEEE, pp. 963–972.

Dastani, M., Riemsdijk, M. B. V. & Meyer, J.-j. C. (2006), Goal types in agent programming, *in* 'In Proceedings of the 17th European Conference on Artificial Intelligence', pp. 220–224.

Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J. C. & Katz, D. S. (2005), 'Pegasus: A framework for mapping complex scientific workflows onto distributed systems', *Scientific Programming* **13**(3), 219–237.

Erdmann, M. (1995), 'Understanding action and sensing by designing action-based sensors', *The International journal of robotics research* **14**(5), 483–509.

Fikes, R. E. & Nilsson, N. J. (1972), 'Strips: A new approach to the application of theorem proving to problem solving', *Artificial intelligence* **2**(3), 189–208.

Gat, E. (1991), 'Integrating reaction and planning in a heterogeneous asynchronous architecture for mobile robot navigation', *ACM SIGART Bulletin* **2**(4), 70–74.

Gat, E. (1993), 'On the role of stored internal state in the control of autonomous mobile robots', *AI magazine* **14**(1), 64.

Gat, E. et al. (1998), 'On three-layer architectures'.

Gerevini, A. E., Haslum, P., Long, D., Saetti, A. & Dimopoulos, Y. (2009), 'Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners', *Artificial Intelligence* **173**(5), 619–668.

Gerevini, A. & Long, D. (2005), 'Bnf description of pddl3. 0', *Unpublished manuscript from the IPC-5 website* .

Ghallab, M., Nau, D. & Traverso, P. (1995), 'Project management: a systems approach to planning, scheduling, and controlling', *New York* p. 356.

Ghallab, M., Nau, D. & Traverso, P. (2004), *Automated Planning: Theory and Practice*, Elsevier.

Kephart, J. O. & Chess, D. M. (2003), 'The Vision of Autonomic Computing', *IEEE Computer* **36**(1), 41–50.

Latombe, J.-C. (1996), 'Robot motion planning, chapter'.

Lee, K., Paton, N. W., Sakellariou, R., Deelman, E., Fernandes, A. A. & Mehta, G. (2009), 'Adaptive workflow processing and execution in pegasus', *Concurrency and Computation: Practice and Experience* **21**(16), 1965–1981.

Lee, K., Paton, N. W., Sakellariou, R. & Fernandes, A. A. (2011), 'Utility functions for adaptively executing concurrent workflows', *Concurrency and Computation: Practice and Experience* **23**(6), 646–666.

Lee, K., Sakellariou, R., Paton, N. W. & Fernandes, A. A. A. (2007), Workflow adaptation as an autonomic computing problem, *in* 'Proceedings of the 2nd Workshop on Workflows in Support of Large-Scale Science', ACM Press, pp. 29–34.

Mahadevan, S. & Connell, J. (1992), 'Automatic programming of behavior-based robots using reinforcement learning', *Artificial intelligence* **55**(2), 311–365.

Mann, A. (2008), Quantitative evaluation of human-robot options for maintenance tasks during analogue surface operations, *in* 'Proceedings of the 8th Australian Mars Exploration Conference', p. 2634.

Marks, M. A., Mathieu, J. E. & Zaccaro, S. J. (2001), 'A temporally based framework and taxonomy of team processes', *The Academy of Management Review* **26**(3), 356.

McCarthy, J. (1963), *Programs with common sense*, Defense Technical Information Center.

Minguez, J., Lamiraux, F. & Montesano, L. (2005), Metric-based scan matching algorithms for mobile robot displacement estimation, *in* 'IEEE International Conference on Robotics and Automation', Vol. 4, Citeseer, p. 3557.

Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B. et al. (2008), 'Junior: The stanford entry in the urban challenge', *Journal of field Robotics* **25**(9), 569–597.

Munson, J. H. (1971), Robot planning, execution, and monitoring in an uncertain environment, *in* 'IJCAI', pp. 338–349.

Newell, A. & Simon, H. A. (1961), *GPS, a program that simulates human thought*, Defense Technical Information Center.

Nilsson, N. J. (1984), Shakey the robot, Technical report, DTIC Document.

Noreils, F. R. & Chatila, R. G. (1995), 'Plan execution monitoring and control architecture for mobile robots', *Robotics and Automation, IEEE Transactions on* **11**(2), 255–266.

Schlenoff, C., Albus, J., Messina, E., Barbera, A. J., Madhavan, R. & Balakirsky, S. (2006), 'Using 4d/rcs to address ai knowledge integration', *AI Magazine* **27**(2), 71.

Schwartz, J. T., Sharir, M. & Hopcroft, J. E. (1987), *Planning, geometry, and complexity of robot motion*, Intellect Books.

Shanahan, M. (2000), Reinventing shakey, *in* 'Logic-based artificial intelligence', Springer, pp. 233–253.

Small, N., Mann, G. & Lee, K. (2013), Goal accomplishment tracking for automatic supervision of plan execution, *in* 'Australasian Conference on Robotics and Automation (ACRA 2013), UNSW, Sydney, Australia'.

Thrun, S., Beetz, M., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Haehnel, D., Rosenberg, C., Roy, N. et al. (2000), 'Probabilistic algorithms and the interactive museum tour-guide robot minerva', *The International Journal of Robotics Research* **19**(11), 972–999.

Winston, P. H. (1984), 'Artificial intelligence', *Reading, Mass.: Ad-dison-Wesley* .