Requirements for Inferring the Intensions of Multitasking Reactive Agents

Michael Freed

NASA Ames Research Center
mfreed@arc.nasa.gov

Eric Dahlman

Colorado State University
dahlman@cs.colostate.edu

Abstract

Multitasking complicates the problem of intent inference by requiring intent inference mechanisms to distinguish multiple streams of behavior and recognize overt task coordination actions.. Given knowledge of an agent's problem representation and architecture it may be possible to disambiguate the agent's actions and better infer its intent. The requirements for implementing such a system are explored within the context of the Apex reactive planner. A prototype reactive inference method based on the Apex architecture is presented.

Introduction

When people persevere overlong at a single task, it is considered pathological. More typically, people switch between tasks, interleaving in order to satisfy task requirements, meet emerging new demands, and take advantage of lulls in one task to make progress at another. Multitasking is the norm in everyday task domains such as cooking and driving a car. It is similarly common in domains such as air traffic control, aircraft operation and nuclear power plant management – areas where technology for inferring human intentions could be usefully incorporated into, e.g. decision-aiding systems and collaborative agents.

Multitasking significantly complicates the problem of intent inference. Two tasks that would easily be recognized if executed separately might appear a meaningless mishmash of behavior when interleaved. Inferring multiple intentions thus requires distinguishing multiple streams of behavior. Multitasking raises other issues as well. For instance, task interleaving often requires an agent to carry out overt task coordination actions – i.e. actions that make it safe to interrupt a task, maintain a task's viability while interrupted, restore its preconditions before resuming, and so on. Failure to distinguish coordination actions from actions on the main goal-path can result in false attribution of their intent and thus difficulty determining the intent of goal-path actions. For example, turning down the stove heat on a pan of pasta before answering the phone might be falsely interpreted as a normal step in a cooking plan (recipe), making it more difficult to match to any known plan. If this action and the later act of turning the heat back up are instead categorized as interruption transitions, matching is facilitated.

Our goal is to incorporate intent inference capabilities into Apex, an agent architecture used to simulate human operators in domains such as air traffic control and commercial jet piloting. Expert performance in these domains requires adapting routine behavior in response to uncertainty and time-pressure. Reactive planners (Firby 1987) have been found to be particularly well suited to meet these requirements (Freed and Remington 1997; John et al. 2002). Expert performance in these domains also demands a relatively sophisticated capability for managing multiple, high-level tasks, a requirement not met by most reactive planners. Apex incorporates a reactive planner extended specifically to meet these multitasking requirements (Freed 1998, 2000).

Since reactive planning has proven a useful basis for modeling human behavior in such domains, we believe it provides a useful conceptual basis for inferring human intent in them as well. Apex, like other reactive planners, generates behavior using a library of stored plans. The planner selects, refines and adapts plans based on conditions at execution time (i.e. reactively). Our approach is to use these stored plans not only to generate behavior, but also to infer other agents' intentions. Observed actions and execution conditions can be matched to stored reactive plans, the goal(s) associated with any matched plan constituting the observed agent's intent. In this paper, we focus on defining the problem and specifying the requirements for a solution based on this approach.

Procedure Representation

Apex represents plan knowledge using Procedure Definition Language, a notation that includes elements specialized for multitask management. The central language construct in PDL is a *procedure*, which contains at least an *index clause* and one or more *step clauses*. The index uniquely identifies the procedure and describes a class of goals the procedure can be used to accomplish. Each step clause describes a subtask or auxiliary activity prescribed by the procedure.

```
(procedure
   (index (cook pasta sauce in ?saucepan))
   (profile hands eyes)
   (step s1 (fetch ?saucepan))
   (step s2 (fetch pasta sauce => ?sauce))
   (step s3 (pour ?sauce into ?saucepan)(waitfor ?s1 ?s2))
   (step s4 (place ?saucepan on stove burner => ?b)
      (waitfor ?s3))
   (step s5 (turn burner ?b to med) (waitfor ?s3))
   (step s6 (stir sauce)
      (period :recurrent :recovery (1 minute))
      (waitfor ?s4 ?s5))
   (step s7 (turn burner ?b to off)
      (waitfor (bubbling sauce)))
   (step end (terminate) (waitfor ?s7))
   (step int (turn burner ?b to low)
      (waitfor (interrupted ?self)))
   (step res (turn burner ?b to med)
   (waitfor (resumed ?self))))
```

Figure 1. Example PDL procedure

The procedure in Figure 1, representing a method for cooking pasta sauce, illustrates several important aspects of PDL. A procedure's steps are not necessarily carried out in the order listed or even in a sequence. Instead, steps are assumed to be concurrently executable unless otherwise specified. If step ordering is desired, a *waitfor clause* is used to specify that the completion (termination) of one step is a precondition for the start (enablement) of another. In the example above, the step labeled *s1* does not contain a waitfor clause and thus has no preconditions; it can begin execution as soon as the procedure is invoked. Steps *s4* and *s5* must wait for *s3* to complete but are not otherwise constrained by the logic of the procedure. They can execute in parallel unless some factor not specified within the scope of the procedure prevents this. For example, if both steps require use of the right hand, one will have to wait for the other to complete. The step labeled *int* specifies what to do if the procedure is interrupted. This step specifies a contingent activity; the procedure can run to completion without ever executing this step.

Procedures are invoked to carry out an agent's active *tasks*. Tasks, which can be thought of as agent goals[1], are stored on a structure called the *agenda* internal to the agent architecture. When a task on the agenda becomes enabled (eligible for immediate execution) what happens next depends on whether or not the task corresponds to a primitive action. If so, the specified action is carried out and then the task is terminated.

If the task is non-primitive, the planner retrieves a procedure whose index clause matches the task. For example, a task of the form *(cook pasta sauce in pan-5)* matches the index clause of the procedure above. The procedure would thus be retrieved once the task became enabled. Step clauses in the selected procedure are then used as templates to generate new tasks, which are then added to the agenda. In this example, there are eight steps so eight new tasks will be created. The process of decomposing a task into subtasks on the basis of a stored procedure is called *task refinement*. Since some of the tasks generated through this process may themselves be non-primitive, refinement can be carried out recursively. This results in the creation of a task hierarchy.

PDL and its interpreter (the reactive planner) support several facets of multitask management including concurrency control, graceful interruption handling and efficient resource scheduling under uncertainty (Freed 1998). A simple example of concurrency control is illustrated by step *s4*, *s5* and *s6* in the above procedure. In this case, the steps are constrained to converge – i.e. *s4* and *s5* can run in parallel but both must complete before *s6* can begin. Other control idioms such as synchronization and racing can also be expressed. The procedure steps labeled *int* and *res* illustrate graceful interruption handling. A simple multitasking approach would treat tasks analogously to processes in an operating system: when interrupted, the system would save state (remember where it left off) but otherwise transition to the new task abruptly. In PDL, it is possible to express interrupt- and resume-time behaviors that minimize undesirable consequences of interruption. The *profile clause* is used to declare resources needed to carry out the task. Integrated scheduling mechanisms automatically detect resource conflicts (i.e. when two enabled tasks require use of the same resource) and invoke priority-based scheduling to resolve the conflict. Moreover, as circumstances change during task execution, priority can be reevaluated, possibly resulting in an ongoing task becoming interrupted to allow execution of a new or newly important task. As a result, Apex agents use limited resources efficiently and adaptively.

---

[1] The term task generalizes the concept of a classical goal – i.e. a well-defined state, expressible as a proposition, that the agent can be seen as desiring and intending to bring about (e.g. "be at home"). Tasks can also, e.g., encompass multiple goals ("be in car seat with engine started and seatbelt fastened"), specify goals with indefinite state ("finish chores"), specify goals of action rather than state ("scan security perimeter "), and couple goals to arbitrary constraints ("be at home by 6pm").

Requirements

The basic process underlying our intent inferencing approach is to match observed events, including agent actions and task environment occurrences, to PDL procedures. Successfully implementing this approach requires considering the information content and semantics of these procedures – i.e. the behavior of the reactive planner that interprets them. Our analysis reveals two main sources of difficulty in using these procedures in their current form. First, agents in realistic task environments will have limited ability to observe match-relevant events. Some events will be unobservable in principle – e.g. mental actions prescribed by steps of an executing procedure. Other important events may go unobserved if the inferring agent needs to share sensor resources with tasks other than observing the performing agent. Moreover, many applications of an intent inference capability, e.g. advising an agent or taking over its tasks when needed, require inferring intent prior to completion of the procedure. Thus intent must be inferred before observing all the actions that could be observed.

With observations providing only incomplete information, exact matches between procedures and observed events cannot be expected. To assess partial matches, inference mechanisms must account for variability in the degree to which given observations discriminate among potential matches – i.e. their diagnosticity. One possibility is to represent diagnostic information (e.g. prior probability that an observed action, action eliciting event, or decision-shaping factor implies the use of a given procedure) directly in the procedure. This is an especially attractive option for routine procedures such as those generally used in reactive planners since the information can be obtained empirically. However, PDL does not currently provide notational elements for representing this kind of information. Thus, one requirement for implementing our approach is to extend PDL for this purpose.

A second kind of challenge to using PDL procedures as the basis of an intent inference approach is the relatively complex semantics of these procedures. In particular, procedures do not fully specify behavior. Actions can be specified abstractly, contingently and with parametric and constraint information whose effect on execution depends on what other tasks are active and what circumstances hold in the task environment. Thus, the relationship between the actions of an Apex agent and underlying procedures can only be understood in terms of the reactive planner that interprets and executes those procedures. Several facets of the reactive-planner's behavior and their implied requirements for intent inference are considered below. The first three, hierarchical decomposition, reactivity and contingency handling, are relevant to reactive planners in general. The latter three, interruption transitions, concurrency control and resource scheduling, are more specific to the way Apex manages multiple tasks.

Hierarchical decomposition Non-primitive steps of a procedure are recursively decomposed into substeps, producing a task hierarchy. In itself, this raises few issues for intent inference since direct observations of action can be handled uniformly with inferred actions. For instance, directly observing an agent pour sauce into a pan can be used to infer the intention of warming up pasta sauce; this in turn can be used to infer the intention of cooking spaghetti. However, two processes occurring during task decomposition imply requirements for our approach. First, variables may become bound. For example, invoking the procedure in Figure 1 binds a variable that specifies a particular saucepan. For inferring intent, variable bindings constitute matching constraints. Second, task decomposition can follow selection between alternative procedures (and thus alternative decompositions). Observing a particular procedure implies constraints on the value of selection criteria expressed in the calling procedure.

Reactivity Reactive procedures include steps that are enabled in response to external events such as the contents of a pan starting to boil. Intent inferencing mechanisms must use observations of such events as evidence for or against procedure matches in the same way it uses observations of agent actions. One important consequence of reactivity is that the order in which steps will be carried out can vary across instances of executing a procedure, as directed by exogenous enabling events. Intent inference mechanisms must therefore be able to match to a space of possible execution paths.

Contingency handling Reactive procedures may include contingent steps – i.e. actions to be carried out only in particular conditions. Since these may or may not occur during a particular instance of executing a procedure, the set of actions constituting a correct execution of the procedure can vary. Inference mechanisms must be able to match to a space of possible action sets.

Concurrency control The steps of PDL procedures are concurrent by default. Concurrency constraints may be expressed in several ways. Waitfor clauses and task control actions (such as task termination) can be used to arrange several control strategies such as fixed ordering, convergence, racing, staggered starts and synchronization. The specified strategy is an important matching constraint for intent inference mechanisms since people often use a range of procedures with common constituent actions but varying ordering strategy.

Interruption transitions People often carry out special actions to minimize the risks and lost progress due to task interruption (e.g. turning down the heat on rapidly heating sauce to avoid a mess). Other actions may be carried out

during the interruption interval to maintain task viability or upon resumption to restore preconditions violated during the interruption interval. Intent inference mechanisms must correctly categorize these as interruption transition actions and not as actions on the main goal-path, otherwise it will be difficult to find a correct match.

Resource scheduling People engage in a range of explicit and implicit behaviors to make multitask behavior more time-efficient. They may interleave tasks that could be carried out serially or take advantage of slack time in the use of resources by one task to make progress at another (e.g. clean up the kitchen while waiting for the sauce to heat). The resulting interleaving of behaviors requires matching mechanisms to decide which actions can be grouped together as serving a common intention. Some actions will unambiguously belong to certain procedures and not others. Others however may be ambiguous (e.g. something may be put in the refrigerator either to store it or cool it down). In these cases, intent inference mechanisms will need to treat the scheduling efficiency behavior of the planner as a constraint on matching. More complex reasoning about this process is required if intent inference mechanisms are employed to check the behavior of other agents and warn of possible error. For example, human air traffic controllers and pilots under high workload sometimes "tunnel" on a task (remain focused on it overlong) or "thrash" (move erratically and rapidly between tasks).

Technical Approach

The intent of a reactive planner like Apex is contained in the set of tasks that are currently being pursued by the planner. The inferred intent of such an agent can be modeled as a set of hypothetical procedures that the agent may be executing. An attractive approach for representing this set in a reactive planner like Apex is through the use of *hypothesis procedures* derived from the PDL procedures controlling the observed agent. An executing hypothesis procedure monitors for events that strengthen or weaken its associated hypothesis and adjust a probability reflecting the likelihood of the hypothesis. This information is directly available to the planner that can then choose to act on the inferred results accordingly.

The driving goal is to meet the requirements in the previous section. An important step toward that end is the automatic synthesis of hypothesis procedures from a PDL plan library into a library of inference procedures. In order to increase the ultimate usefulness of intent inference in Apex a knowledgeable user should be able to include extra advice about the domain. This provides and avenue to include inference information that may not be expressed in the domain or effectively inferred by the default approach.

As an example lets consider a procedure for recognizing the *cook pasta sauce* task based on the actions observed. To simplify this example we will assume our observing agent has a perfect sensor that will create events like *(observed stir sauce)* for the visible actions performed by the other agent. Given these assumptions a potential recognition procedure can be seen in Figure 2.

The approach we used to derive these recognition procedures is to analyze the graph of potential step transitions within the procedure. As the agent perceives new events these may indicate transitions within this graph and the associated procedure. Because there may be several active tasks capable of generating the event the probability that the event did belong to that task must also be calculated. The values for the example in Figure 2 followed from the fact that there were 10 recipes that made use of a saucepan in the plan library while only one of them was for pasta sauce.

While runtime probability calculations may be more accurate precalculated values open up the possibility simplifying the resulting hypothesis procedures. We can see the effect of this simplification on the pasta sauce recognition procedure where there will not need to be any checks for *(observed stir sauce)* since these observations will not have an effect on the inferred probability in the case. In noisier domains where we need to contend with errors in sensors and partial observability it may not be beneficial to discard these events as they would help strengthen the *make pasta sauce* hypothesis.

Once the transition graph for a procedure has been generated the associated hypothesis procedures can be created. There is some flexibility in how this can be done but the general approach is to make the hypothesis procedure's index be a predictive event. For instance, in the case of making pasta sauce stirring the sauce is a highly predictive event. However, that event may occur too late in the process for the inference to be useful to an agent who is allergic to tomatoes. There is also a danger in selecting the first event as the trigger since the agent may become overloaded with too many "fanciful" hypotheses.

```
(procedure ;; Cooking pasta sauce?
   (index (observed fetch saucepan))
   (step s1 (set-probability (cook pasta sauce) 0.1))
   (step s2 (set-probability (cook pasta sauce) 1.0)
      (waitfor (observed fetch pasta sauce)))
   (step done (set-probability (cook pasta sauce) 0.0)
      (waitfor (observed turn burner ?b to off)))
   (step end (terminate) (waitfor ?done))
   (step interrupted
      (set-probability (interrupted cook pasta sauce)
            (probability (cook pasta sauce)))
      (waitfor (observed turn burner ?b to warm)))
   (step resumed
      (set-probability (interrupted cook pasta sauce)
            0.0)
      (waitfor (observed turn burner ?b to hi))))
```

Figure 2. An example hypothesis procedure.

Future Work

The problem of embedding intent inference into the Apex reactive planner requires further examination before the full requirements are understood. The hypothesis procedure approach suggested here appears promising in that it meshes well with the reactive planning paradigm and provides useful information in several toy examples. Currently, however, only the most elementary types of evidence, observed events and procedure structure, are considered when generating the hypothesis procedures. Although this allows successful intent inference in some cases, a great deal more information about the task execution context, task environment regularities and procedure semantics could still be employed improve inference accuracy. Our examples indicate numerous open questions such as the need to consider how much effort can and should be dedicated to inferring the intent of other agents in the environment. Currently there is no concept of the utility of making an inference this need to be captured and possibly even allowing the agent to adjust it to the situation at hand. In future work we will assess the effectiveness of our approach with more rigorous evaluation on larger more involved domains.

References

Engelmore, R., and Morgan, A. eds. 1986. *Blackboard Systems*. Reading, Mass.: Addison-Wesley.

Firby, R.J. 1989. Adaptive Execution in Complex Dynamic worlds. Ph.D. thesis, Yale University.

Freed, M. (2000) Reactive Prioritization. *Proceedings 2nd NASA International Workshop on Planning and Scheduling for Space*. San Francisco, CA.

Freed, M. (1998a) Simulating Human Behavior in Complex, Dynamic Environments. Doctoral Dissertation. Department of Computer Science, Northwestern University.

Freed, M. (1998b) Managing multiple tasks in complex, dynamic environments. In *Proceedings of the 1998 National Conference on Artificial Intelligence*. Madison, Wisconsin.

Freed, M. and Remington, R. (1997) Managing decision resources in plan execution. *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Nagoya, Japan.

Gat, Erann. 1992. Integrating planning and reacting in heterogeneous asynchronous architecture for controlling real-world mobile robots. In *Proceedings of 1992 National Conference on Artificial Intelligence*.

John, B. E., Vera, A. H., Matessa, M., Freed, M., and Remington, R. (2002) Automating CPM-GOMS. In *Proceedings of CHI'02: Conference on Human Factors in Computing Systems*. ACM, New York, pp. 147-154.