Implementing a Task-Oriented Time-Situated Agent

Darsana P. Josyula DARSANA@CS.UMD.EDU
Anthony Herron HERRONA0814@STUDENTS.BOWIESTATE.EDU
Kenneth M'Bale KMBALE@GMAIL.COM
Department of Computer Science, Bowie State University, Bowie, MD 20715 USA

Abstract

Real-time agents with a limited set of primitive actions produce complex behaviors by choosing the parameters and timing of the actions appropriately. These parameters and timings are often based on plans that are specified or learned. The planning process is usually separate from the anomaly and goal management process and hence managing interactions between the two processes become complex. This position paper discusses how task-specific planning knowledge and task-general knowledge for anomaly and goal management can interact in the time-sensitive Active logic framework to produce, monitor and adjust task-oriented agent behaviors in real-time.

1. Introduction

The behavior of task-oriented, goal-directed agents tends to rely on explicit sophisticated plans generated by planning systems. Presented with a task, a planning system deliberates to choose a plan that has been learned or stored. Having chosen a plan, a goal-driven agent executes the plan while simultaneously optimizing the execution to meet the agent's goals. For real-time agents, the conditions that render a plan viable may no longer hold as the agent performs a task. Additionally, since time is not at a standstill during agent deliberation, the time spent on planning affects the time available for performing actions. Hence, checking preconditions, adopting goals, detecting conflicts, and choosing actions, all need to be interleaved in time for agents to remain responsive to changes and exhibit effective real-time behavior.

Though managing active goals dynamically is a core foundation of goal-driven autonomy (Muñoz-Avila, 2018; Molineaux et al., 2010), implementations (Weber et al., 2010; Beetz, 2002; Shivashankar et al., 2013) often separate the planner from the controller that manages goals and anomalies. This leads to complex plan-repair and re-planning (Coddington, 2006; Dannenhauer & Munoz-Avila, 2015) when conditions change. Real-time agents situated in dynamically changing environments cannot afford complicated plan-repair and re-planning while remaining responsive to the anomalies that occur in their environments. Interleaving learning, planning and execution can produce real-time behaviors (Jaidee et al., 2013, Ghallab et al., 2016).

In this position paper, we discuss a methodology based on a time-situated framework – Active logic (Elgot-Drapkin & Perlis, 1990) – that integrates the planning, anomaly processing and goal management within the same reasoning process. The methodology uses a hierarchical goal structure to represent the agent's task-specific knowledge. The goal hierarchy is akin to the task structures that planners based on hierarchical task networks (Georgievski & Aiello, 2015) make use of. The goal hierarchy gives a structural decomposition of how a task-request can be decomposed into

goals, subgoals and actions that the agent can initiate on its own. The methodology uses a set of control rules that manage anomalies and goals to provide real-time behaviors in accordance with changes in the environment and the passage of time.

2. Active Logic and Task-Oriented Agency

We use *task* to denote the activity assigned to an agent by an external agent (human). The agent adopts *goals* internally to perform tasks. Thus, the task of retrieving a book from the living room may result in adopting a goal to move to the living room and another goal to locate the book. For real-time behavior, an agent must keep track of its progress towards these goals and respond to changes as time and situations evolve.

With reasoning proceeding temporally in a step-by-step manner, Active logic (Perlis et al., 2017) provides a mechanism to integrate changes in the environment, passage of time and progress towards goals into the ongoing inferences. Active logic's ability to detect contradictions and distrust contradictory formulas helps to manage conflicting information (e.g., contradictory goals and prerequisites). The non-monotonic nature of Active logic allows formulas derived in one step to be removed or distrusted in a later step when they no longer hold. Thus, without complex re-planning, an agent can adopt goals as their prerequisites become true in the environment and drop them as they are fulfilled or become unnecessary or unfulfillable.

Active logic allows defining rules for observation processing, goal formation, action selection, and anomaly detection as data that can be manipulated the same way as the task-specific planning knowledge. This means that Active-logic based agents have the potential to not only exhibit real-time behavioral changes when task contexts or environments change but they can also cause changes to the control processes for anomaly and goal management in real-time. Active-logic based reasoners (Purang, 2001) have been previously used to provide interleaved planning and execution (Nirkhe, 1995) in task-oriented agents (Josyula, 2005). This paper extends the previous framework with task-specific goal hierarchy structures to manage more complex tasks.

3. Schema for Task Specifications

We define a general schema for specifying task-related knowledge as a goal hierarchy similar to the hierarchical goal network (Shivashankar et al., 2013), within the Active logic framework. The top level of the goal hierarchy is the task and the bottom level has the primitive actions that the agent can perform. Complex and primitive goals form the internal nodes in the hierarchy. We use the following notation to define the schema:

- *T* denotes the set of tasks and *G* denotes the set of goals that can lead to task completion.
- A lower-case letter denotes an element of the set denoted by its equivalent upper-case letter.
- Goals are either primitive *PG* or complex *CG*, *i.e.*, $G = CG \cup BG$. A complex goal is made of subgoals which may be primitive or complex, whereas a primitive goal is made exclusively of actions the agent can execute. Each complex goal $cg \in CG$ is a set, where $cg = \{x \mid x \in G\}$. Each complex goal may be an ordered or an unordered set.
- *A* is the set of action predicates associated with the primitive actions of the agent.

We now define the task specification schema as *TSS* = {*P, AG, GG, R, S, U, V, Z*} for each task, where each of the components are as follows:

- P is the purpose of a task, where $P: T \rightarrow G$.
- AG is a mapping from primitive goals to an action, where $AG: PG \rightarrow A$.
- GG shows the subgoal composition of each goal, where $GG: CG \rightarrow G$, is a mapping from a complex goal to another primitive or complex goal.
- R specifies the preconditions for actions, where $R: A \rightarrow F$ and $a \in A$ is mapped to $f \in F$ if the agent must know f before it can execute the action associate with a.
- S specifies the preconditions for goals, where $S: G \rightarrow F$ and $g \in G$ is mapped to $f \in F$, if the agent must know f before it can adopt the goal g.
- U denotes the post victory conditions that indicate the success of actions, where $U: A \rightarrow F$ and a is mapped to f if f is a general template for the expectation e that must hold if a succeeds.
- V denotes the post victory conditions that indicate the success of goals, where $V: G \rightarrow F$ and g is mapped to f if f is a general template for a condition that must hold if g succeeds.
- Z denotes the optimization conditions for actions, where $Z: A \rightarrow F$ and $a \in A$ maps to $f \in F$ if an instantiation of f provides parameter values to optimize the action a.

4. Active-logic based Rule Definition Language

The control rules for agent behavior are specified in a generalized first order language based on Active logic. The relevant parts of the language necessary to understand the rules specified in Section 5 is described next.

Forward-if rules: These are generalized modus ponens rules with the caveat that the consequent is derived one step after all the antecedents become true. Once a consequent is derived, it is added to the KB and takes part in other subsequent inferences.

Introspection predicates: Special predicates for introspection used in the rules and their explanation are given below in Table 1.

Table 1: Introspection predicates

$inKB(K)$	True if an instantiation of *K* exists in the *KB; False* otherwise
$\sim inKB(K)$	*True* if no instantiation of *K* exists in the *KB; False* otherwise
$finKB(L)$	*True* if $\forall\ l \mid l \in L$ and an instantiation of *l* exists in the KB; *False* otherwise
$\sim sinKB(L)$	*True* if $\exists\ l \mid l \in L$ and an instantiation of *l* does not exist in the *KB*; *False* otherwise

5. Task General Control Rules

The key control rules for goal formation and anomaly management are listed. Predicates start with lowercase. Variables start with uppercase. Comma between predicates indicates the *and* operation.

1. If the current time is *N* and current task is *T*, then note the time that the task started.
   $task(T),\ now(N) \rightarrow taskStarted(T, N)$.
2. If the purpose of task *T* is goal *G* and success conditions V of goal G are not met, then form goal G
   $task(T), purpose(T, G), success(G, V), \sim sinKB(V) \rightarrow initiateGoal(G)$
3. Adopt subgoals to accomplish current goal.

$goal(G), subgoal(G, SG), success(SG, V), \sim sinKB(V), prereq(G, S), finKB(S) \rightarrow initiateGoal(SG)$.

4. If the preconditions for the goal are known, then assert goal.
   $initiateGoal(G), prereq(G, S), finKB(S) \rightarrow goal(G)$.
5. Attempt actions to accomplish a goal if the agent is not currently doing another action.
   $goal(G), action(G, A), \sim inKB(doing(A1)), prereq(A, R), finKB(R), optimize(A, Z), finKB(Z) \rightarrow do(A)))$.
6. Mark goals as fulfilled if their success conditions are met.
   $goal(G), success(G, V), finKB(V), now(N), \sim inKB(fulfilled(G, V, N1)) \rightarrow fulfilled(G, V, N)$.
7. Drop the goals that have an associated fulfilled goal.
   $goal(G), fulfilled(G, V, N) \rightarrow drop(goal(G))$.
8. Once the success conditions of a task are met, note that the task is completed.
   $task(T), purpose(T, G), success(G, V), finKB(V), now(N), \sim inKB(taskDone(T, N1)) \rightarrow taskDone(T, N)$.
9. Drop tasks that are completed.
   $task(T), taskDone(T, N) \rightarrow drop(task(T))$.
10. Once a task deadline has elapsed, no need to hold on to goals associated with that task.
    $overtimeAt(T, N), purpose(T, G), goal(G) \rightarrow drop(goal(G))$.
11. When actions are initiated, create associated expectations.
    $success(A, U), doing(A) \rightarrow expect(A, U)$.
12. If an expectation for action fails, then note that the action failed.
    $done(A), expect(A, U), \sim inKB(U) \rightarrow failed(A)$.

6. Agent Architecture

This section presents the architecture of an agent based on the framework described above. The environment that the agent is situated in has objects and other agents that it can perceive. An agent has two methods – *do()* to perform actions and *get()* to receive observations. Agent reasoning and behavior proceeds by querying and modifying the agent's current KB as illustrated in Figure 1.

An agent's KB has several parts. The Object Knowledge Base (OKB) stores the information about the objects that the agent perceives. The Task Knowledge Base (TKB) stores the definitions of the tasks that the agent can perform as goal hierarchies using the task specification schema (TSS) defined in Section 3. The Rules Knowledge Base (RKB) stores the task general rules of control defined in Section 5. In Figure 1, RKB forms the unshaded shapes and arrows. The Derived Knowledge Base (DKB) stores formulas that are derived by applying inferences using rules in the RKB. The Expectations Knowledge Base (EKB) is a subset of DKB that stores formulas that are expected to become true when an action succeeds. The Performance Knowledge Base (PKB) is a subset of TKB and it stores optimization parameters that can improve action performance.

The agent receives a task request through its interface. The agent queries the TKB to determine if it knows how to perform the task. A task is mapped to a main goal through the task's purpose. If the TKB contains the goal-hierarchy definition of the task, the agent identifies the purpose of the task and applies Rule 2 to initiate a goal. If the TKB does not contain sufficient information about the requested task, the agent asks for help. A goal (complex) is composed of other goals and actions. The agent consults its TKB and OKB to select appropriate actions to fulfill its goals. An action invokes the agent's do() method with the appropriate action parameter to perform that action. As an action is initiated, the agent uses Rule 11 in its RKB to project the expected state of the OKB or DKB once the action is done. The expectation is added to EKB at the next time step. The agent's

get() method allows it to perceive the result of an action as a change to one or more objects that it keeps track in its OKB. The agent compares its current DKB and OKB contents with that in the EKB to note expectation violations and failed actions.

A learner takes the contents of OKB and *optimize* templates from the TKB to provide optimizing parameters for agent actions. These optimizing parameters that are produced by the learner are stored in PKB. The agent uses the information from PKB as it applies Rule 5 in the RKB to invoke an action. In this architecture, learning proceeds hand-in-hand with reasoning and acting, and hence changes in OKB and DKB can automatically change the optimizing parameters for different actions in the PKB.

Contradictions that occur anywhere in the Active logic KB are immediately noted and the corresponding formulas are distrusted. Also, OKB evolves with the agent's current perceptions by removing formulas that no longer hold true. Since all these changes become part of the evaluation in Active logic to determine the formulas that hold true in the next time step, the resultant agent has the potential to be responsive to real-time changes in the environment.

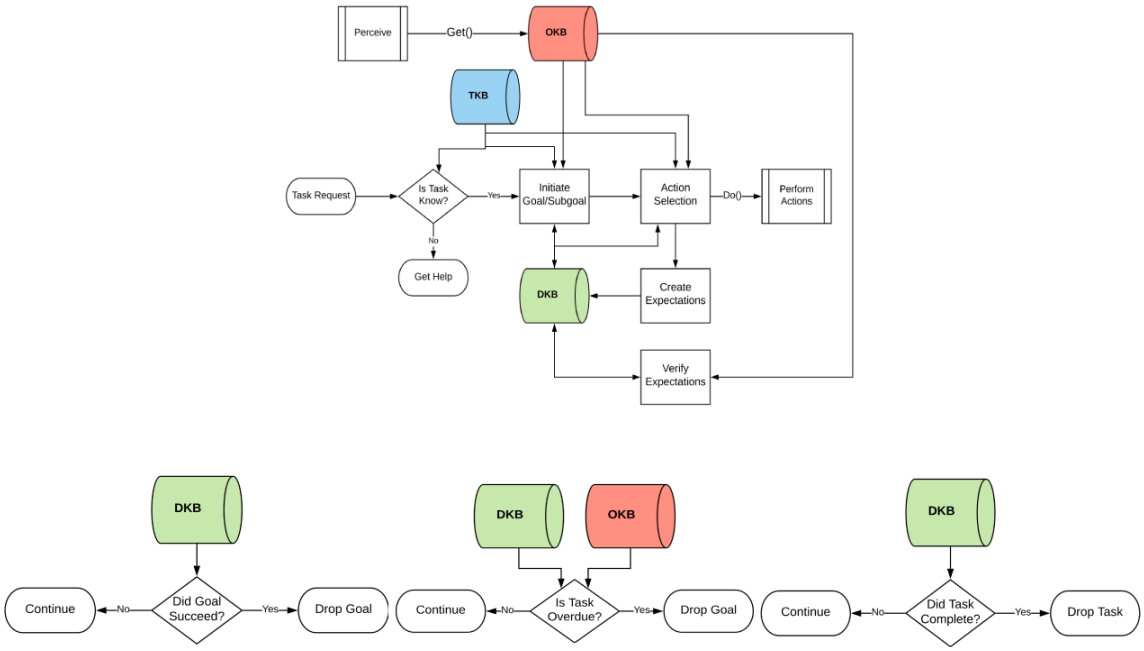

Figure 1: Agent Architecture (Unshaded shapes and arrows form RKB)

7. Conclusion

Agents need general-purpose methods that can produce real-time behaviors for them to work outside of laboratory settings in a variety of environments with different task contexts. We discussed an approach to specify task-general control knowledge for managing goals and anomalies as well as task-specific hierarchical knowledge in a unified time-sensitive Active logic framework that allows easy manipulation of both the control mechanism and task-specific knowledge to produce real-time behaviors. The approach also presents preliminary work on how optimization templates can be specified to integrate learned parameters with the ongoing reasoning for optimizing action performance. The approach provides agents the ability to respond to changes in

the environment, integrate new optimizations learned from experience for performing actions, and handle contradictory information, in real-time.

Acknowledgements

This work has been supported by a grant from DARPA, under Agreement No. HR00112090025.

References

Beetz, M. (2002). *Plan-Based Control of Robotic Agents*. Springer-Verlag Berlin Heidelberg.

Coddington, A. (2006). Motivations for MADbot: A Motivated and Goal directed Robot. In *Proceedings of the 25th Workshop of the UK Planning and Scheduling Special Interest Group*, (pp. 39–46).

Dannenhauer, D., & Munoz-Avila, H. (2015). Raising expectations in GDA agents acting in dynamic environments. In *Proceedings of the 24th International. Conference. on Artificial Intelligence (IJCAI)*, (pp. 2241-2247).

Elgot-Drapkin, J. J., & Perlis, D. (1990). Reasoning situated in time I: basic concepts. *Journal of Experimental & Theoretical Artificial Intelligence*, *2*(1), (pp. 75–98).

Georgievski, I., & Aiello, M. (2015). HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, *222*, (pp. 124–156).

Ghallab, M., Nau, D., & Traverso, P. (2016). *Automated Planning and Acting*. Cambridge University Press.

Jaidee, U., Muñoz-Avila, H., & Aha, D. W. (2013). Case-Based Goal-Driven Coordination of Multiple Learning Agents. In *International Conference on Case-Based Reasoning*, (pp. 164–178).

Josyula, D. (2005). *A Unified Theory of Acting and Agency for a Universal Interfacing Agent*. Ph.D. Thesis, University of Maryland, College Park, MD.

Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-driven autonomy in a navy strategy simulation. In *Proceedings of AAAI*, (pp. 1548–1553).

Muñoz-Avila, H. (2018). Adaptive Goal Driven Autonomy. In *International Conference on Case-Based Reasoning*, (pp. 3–12).

Nirkhe, M. V. (1994). Time-situated Reasoning Within Tight Deadlines and Realistic Space and Computation Bounds. In *AAAI'94: Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence* (pp. 1480). Seattle, Washington, USA.

Perlis, D., Brody, J., Kraus, S., & Miller, M. (2017). The Internal Reasoning of Robots. In *Thirteenth International Symposium on Commonsense Reasoning*.

Purang, K. (2001). Alma/Carne: Implementation of a time-situated meta-reasoner. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence. (ICTAI) 2001* (pp. 103–110).

Shivashankar, V., Alford, R., Kuter, U., & Nau, D. (2013). Hierarchical Goal Networks and Goal-Driven Autonomy: Going where AI Planning Meets Goal Reasoning. In *2013 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning*, (pp. 95–110).

Weber, B. G., Mateas, M., & Jhala, A. (2010). Case-Based Goal Formulation. In *Proceedings of the AAAI Workshop on Goal-Driven Autonomy*.