

Goal Reasoning to Coordinate Robotic Teams for Disaster Relief

Mark Roberts¹, Swaroop Vattam¹, Ronald Alford²,
Bryan Auslander³, Tom Apker⁴, Benjamin Johnson¹,
and David W. Aha⁴

¹NRC Postdoctoral Fellow; Naval Research Laboratory, Code 5514; Washington, DC

²ASEE Postdoctoral Fellow; Naval Research Laboratory, Code 5514; Washington, DC

³Knexus Research Corporation; Springfield, VA

⁴Navy Center for Applied Research in Artificial Intelligence; Naval Research Laboratory, Code 5514; Washington, DC

^{1,2}first.last@nrl.navy.mil | ³first.last@knexusresearch.com | ⁴first.last@nrl.navy.mil

Abstract

Goal reasoning is a process by which actors deliberate to dynamically select the goals they pursue, often in response to notable events. Building on previous work, we clarify and define the Goal Reasoning Problem, which incorporates a Goal Lifecycle with refinement strategies to transition goals in the lifecycle. We show how the Goal Lifecycle can model online planning, replanning, and plan repair as instantiations of Goal Reasoning while further allowing an actor to select its goals. The Goal Reasoning Problem can be solved through goal refinement, where constraints introduced by the refinement strategies shape the solutions for successive iterations. We have developed a prototype implementation, called the Situated Decision Process, which applies goal refinement to coordinate a team of autonomous vehicles to gather information soon after a natural disaster strikes. We outline several disaster relief scenarios that progressively require more sophisticated responses. Finally, we demonstrate the prototype Situated Decision Process on the simplest of our scenarios, showing the merits of applying goal refinement to disaster relief.

1. Introduction

Robotic systems often commit to actions to achieve some goal. For example, a robot may commit to actions that attain some goal (e.g., to be at(location)) or to maintain some desirable condition (e.g., keep its battery charged). Robots also frequently act in partially-observable, dynamic environments with non-deterministic action outcomes. Consequently, robots may encounter notable events that impact their current commitments, examples of which include an exogenous event in the environment (e.g., wind disrupts vehicle navigation), a sensor reading identifies something of interest (e.g., a radio sensor reports the cell phone signal of an important person), or an executed action leads to an unanticipated outcome (e.g., a vehicle switches itself to a more urgent task, causing delay on the first task).

Robots must deliberate on their responses to notable events that impact their goals. Appropriate responses might include continuing despite the event, adjusting expectations, repairing the current plan, replanning, selecting a different goal (i.e., regoaling), deferring the original goal in favor of another goal, or dropping the goal altogether. Responses could be designed a priori or learned by the robot, but ultimately, the robot deliberates about its commitment(s) to its goal(s). *Goal Reasoning* (GR) is the capacity of an actor to deliberate about its goals, which involves formulating, prioritizing, and adjusting its goals during execution. GR actors are distinguished by their available responses, how they obtained them, and how they apply them. The degree to which an actor performs GR determines its autonomy and ability to respond to change.

We have implemented a software library for Goal Reasoning that we apply to coordinating robotic vehicle teams for Foreign Disaster Relief (FDR) operations. In the rest of the paper, we introduce FDR (§2) and our prototype system called the Situated Decision Process (SDP) (§3). We formally extend the GR Problem to online planning, demonstrating how it instantiates common systems (§4). We describe how to solve Goal Reasoning as iterative *Goal Refinement* (§5). We outline a set of FDR scenarios (§6) and detail how we applied our Goal Refinement library for the simplest of the FDR scenarios (§7). After a proof-of-concept demonstration (§8), we conclude and highlight ongoing and future work (§9). Related work is mentioned throughout the sections. The contributions of this paper over previous work (Roberts et al. 2014, 2015) include formally incorporating Goal Refinement into an online planning and execution framework (Nau 2007) and introducing an algorithm, fully outlining the set FDR scenarios, and providing richer detail of the implementation of Goal Refinement for FDR.

2. Motivating Application: Disaster Relief

We study how to coordinate a team of robotic vehicles for Foreign Disaster Relief (FDR) operations. Between the time of a tragic disaster (e.g., Typhoon Yolanda) and the arrival of support operations, emergency response personnel need information concerning the whereabouts of survivors, the condition of infrastructure, and suggested ingress and evacuation routes. Current practice for gathering this information relies heavily on humans (e.g., first responders, pilots, drone operators). A team of autonomous vehicles with sensors can facilitate such information gathering tasks, freeing humans to perform more critical tasks in FDR operations (Navy 1996). It is not tenable to tele-operate every vehicle, so we must design a system that allows humans to be “on” the control loop of vehicles without issuing every vehicle command. FDR operations present unique challenges for domain modeling because each disaster is distinct. Any system that supports FDR operations must allow personnel to tailor vehicles’ tasks to the current situation. The system must also respond to notable events during execution.

An example information gathering task is shown in Figure 1 (top), which depicts a survey task for a team of fixed-wing aerial vehicles. Three vehicles (V1, V2, and

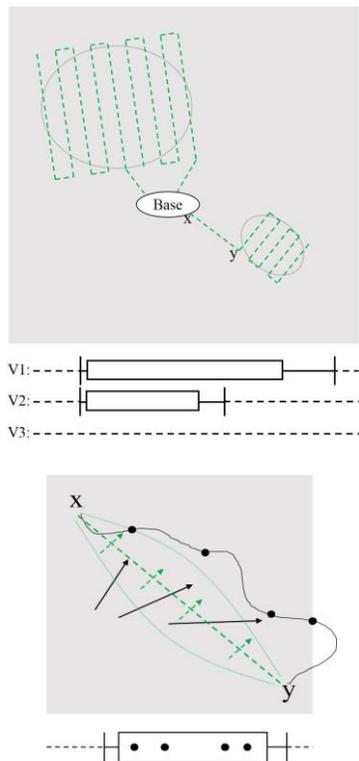


Figure 1: Examples where goal reasoning may apply in a team survey task (top) and a track following task from point x to point y (bottom) with possible notable events highlighted by dots .

V3) begin at the center base and must follow the nominal trajectories (green dashed lines) as closely as possible to maximize coverage of the areas (gray circles). The outer gray box outlines the vehicles’ allowed flight envelope.

Figure 1 (bottom) demonstrates notable events for a single vehicle with a next goal of at(y). The vehicle should follow the expected path (dashed line from x to y) within the preferred bounds (the curved thin green lines); staying within these bounds gives the best solution. The gray outer box is the proposed flight envelope outside of which the vehicle may negatively interact with other vehicles. The actual flight path is given by the solid arc that starts at x and ends at y. The deviating path is due to the difference between the expected wind (dashed vectors) and actual wind (solid vectors). The dots correspond to notable events that could impact the vehicle’s goal commitment to be at(y). The first two points indicate where the vehicle violates the preferred trajectory while the last two points indicate the eminent and actual violation of the flight envelope.

Below each plot is a representation of the vehicle timeline(s), as described by Smith et al. (2000). The time window of the plan indicates that the plan should start executing no earlier than the earliest start time (i.e., the leftmost vertical bar) and finish by the latest finish time (the rightmost vertical bar). The large block in the middle indicates the expected duration.

3. The Situated Decision Process (SDP)

We have implemented a prototype of a system, called the Situated Decision Process (SDP), which is designed to allow flexible assignment and control of a team of robots for FDR. Figure 2 displays an abstraction of the SDP components we discuss in this paper. The SDP is partitioned into three abstract layers, each composed of components that perform specific tasks. We will briefly describe the layers and some components. A more complete exposition of the SDP and its components is provided by Roberts et al. (2015).

The *UI Layer* (colored white) manages interaction with the Operator. In this layer, the User Interface (UI) component collects operational goals and constraints from a human Operator (e.g., survey this region, look for a Very important Person (VIP) in this other region, and do not fly outside these bounds). The UI Layer conveys Operator feedback to the other components as needed and provides info to an Operator that the Operator may then decide to act on.

The *Distributed Layer* (colored black) manages the vehicles or vehicle simulation. Reactive robotic controllers often employ FSAs to determine a robot’s next action. Although they are fast to execute, hand-writing FSAs is



Figure 2: An abstract view of the Situated Decision Process (SDP). Nodes are colored by layer: UI (white), Coordination (Gray), and Vehicle (black).

error prone, tedious, and brittle. Yet, creating a single robotic controller for the many FDR missions and tasks is untenable because no controller could incorporate all the necessary steps. Recent advances apply a restricted variant of Linear Temporal Logic (LTL) called General Reactivity(1) to automatically synthesize FSAs in time cubic in the size of the final FSA (Bloem et al. 2012). This layer leverages LTLMop (Kress-Gazit et al. 2009) for LTL synthesis and physicomimetics (Apker et al. 2014) to implement vehicle control.

The *Coordination Layer* (colored gray) focuses on the mission and task abstractions for the vehicle teams. Even though LTL improves the consistency and speed of FSA generation, synthesis still becomes impractical for teams in dynamic environments. Hierarchical mission planning is naturally suited to limit the FSA size for teams of vehicles (e.g., by pre-allocating missions to vehicles or by assigning vehicles to teams). Assigning specific tasks to vehicles leads to compact, manageable LTL specifications, which allows us to construct vehicle FSAs with reasonable computational effort. We employ hierarchical decomposition (task) planning because it matches well with how humans view FDR operations (Navy 1996). In particular, we apply goal refinement to coordinate those vehicle missions in support of larger FDR operations.

The Coordination and Distributed Layers of the SDP are linked via a set of *Coordination Variables*, which integrate team mission goals with the vehicle controllers by providing abstraction predicates for vehicle commands, vehicle state (e.g., current behavior and health), and abstract vehicle sensor data.

The responses of the SDP must consider relevance to the operational context. Much is unknown or dramatically different from before to the disaster. The SDP must respond appropriately to the Operator dynamically (re)allocating resources or (re)prioritizing goals as new information becomes available. The SDP should respond by confirming the Operator’s intent and producing alternatives that best allocate resources to goals.

4. Goal Reasoning

Deliberating about objectives – how to prioritize and attain (or maintain) them – is a ubiquitous activity of all intentional entities (i.e., actors). For the purposes of this section, we make the simplifying assumption that an objective is a *goal*, which is a set of states the actor desires to attain or maintain. Thangarajah et al. (2011) and Harland et al. (2014) show that all goals are either attainment goals or maintenance goals, but for further simplicity we will focus almost exclusively on attainment goals in this paper. Regardless of the source, achieving goals requires deliberation on the part of the actor (e.g., a plan must be created to achieve a goal). Although our motivating application is robotic team coordination, we generally refer to any system that interleaves deliberation with acting as an *actor*, following the terminology of Ghallab et al. (2014) and Ingrand & Ghallab (2014). This section extends and clarifies early work on formalizing GR by Roberts et al. (2014).

To clarify the relationship of GR to planning, consider our adaptation of Nau’s (2007) model of online planning and execution in Figure 3, which shows how a Goal Reasoner complements online planning (in black) with *Goal Reasoning* (in gray). The world is modeled as a State Transition System $\Sigma = (S, A, E, \delta)$ where: $S = \{s_0, s_1, s_2, \dots\}$ is a set of (discrete) states that represent facts in the world; $A = \{a_1, a_2, \dots\}$ are the actions controlled by the actor; $E = \{e_1, e_2, \dots\}$ is a set of events not controlled by the actor; and, $\delta : S \times (A \cup E) \rightarrow 2^S$ is a state-transition function. $s_{init} \in S$ denotes the initial state of the actor. Assuming attainment goals, the actor seeks a set of transitions from s_{init} to either a single goal state $s_g \in S$ or

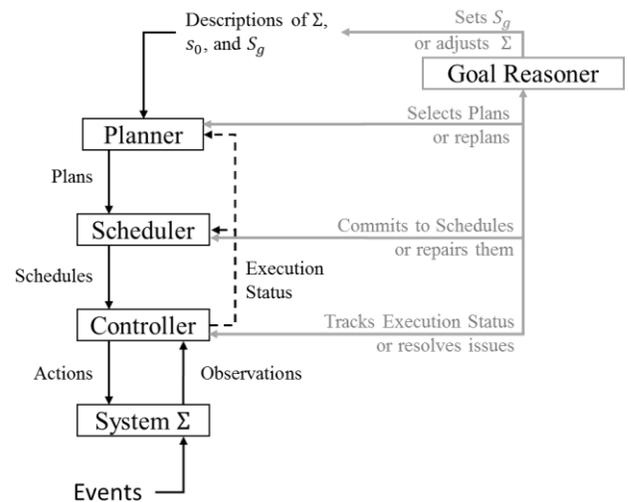


Figure 3: Incorporating Nau’s (2007) Online Planning Model into a Goal Reasoning Loop.

a set of goal states $S_g \subset S$. Under classical assumptions, the planning problem can be stated: Given $\Sigma = (S, A, \delta)$, s_{init} , and a set of goal states S_g find a sequence of actions (a_1, a_2, \dots, a_k) that lead to a sequence of states $(s_{init}, s_1, s_2, \dots, s_k)$ such that $s_1 \in \delta(s_{init}, a_1)$, $s_2 \in \delta(s_1, a_2)$, \dots , $s_k \in \delta(s_{k-1}, a_k)$, and $S_g \in s_k$. However, it is rare that plans exist without some actor to execute them (cf. Pollack & Horty 1999). In *online planning*, execution status is provided to the planner as part of its deliberation. This allows the planner to adjust to dynamic events or new state in the environment. The online planning model often assumes static, external goals.

Before we define Goal Reasoning, we must clarify the notion of “state” in the GR actor, which includes *both* its external and internal state and which requires expanding the state representation beyond S . To avoid confusion with the use of the word *state* as it is typically applied in planning systems, we will use L to represent the *language* of GR. We say that the language of a GR actor is $L = L_{external} \cup L_{internal}$, where

- $L_{external}$ will often be a model of Σ but may be (or may become) a modified or incomplete version of Σ during execution or deliberation. An example of an external state for Figure 1 is at(y).
- $L_{internal}$ represents the predicates and state required for the refinement strategies (e.g., the predicates *attain*(g) or *maintain*(g), the state of all goals). An example of an internal state for Figure 1 is *attain*(at(y)).

We similarly extend and partition the set of goals into $L_g = External_g \cup Internal_g$. In $L_{external}$ the actor selects actions to achieve $External_g$. In $L_{internal}$ the actor selects actions to achieve $Internal_g$. Internal goals may be conditioned on external goals or vice versa. For convenience, we write goals as g and it should be clear from context whether we mean $g \in 2^S$ or $g \in 2^L$. If more context is needed we will use S_g for goals that depend on Σ and L_g for goals that depend on Z (defined next).

We model the GR actor as a State Transition System $Z = (M, R, \delta_{GR})$, where: M is the goal memory that we detail in §4.1; R is the set of refinement strategies introduced in §4.2; and $\delta_{GR} : M \times R \rightarrow M'$ is a transition function we describe in §4.3.

4.1 The Goal Memory (M)

The Goal Memory M stores m goals. Let g_i be the actor’s i^{th} goal for $0 \leq i \leq m$. Then $N^{g_i} = (g_i, parent, subgoals, C, o, X, x, q)$ is a goal node where:

- g_i is the goal that is to be achieved (or maintained);
- $parent$ is the goal whose subgoals include g_i ;
- $subgoals$ is a list containing any subgoals for g_i ;

C is the set of constraints on g_i . Constraints could be temporal (finish by a certain time), ordering (do x before y), maintenance (remain inside this area), resource (use a specific vehicle), or computational (only use so much CPU or memory).

o is current lifecycle mode (see Figure 4 and §4.2).

X is a set of expansions that will achieve the goal. The kind of expansions for a goal depend on its type. For goals from Σ , expansions might be a plan set Π . But other goals might expand into a goal network, a task network, a set parameters for flight control, etc. The **expand** strategy, described in §4.2, creates X .

$x \in X$ is the currently selected expansion. This selection is performed with the **commit** strategy.

q is a vector of one or more quality metrics. For example, these could include the *priority* of a goal, the *inertia* of a goal indicating a bias against changing its current mode because of prior commitments, the net value (e.g., cost, value, risk, reward) associated with achieving g_i , using the currently selected expansion $x \in X$, the parallel execution time (i.e., the schedule makespan) or the number of plan steps.

The constraints will be discussed in §5, where we detail how a GR actor refines goals. A partition $C = C^{provided} \cup C^{added}$ separates constraints into those provided to the GR process independent of whatever invoked it (e.g., human operator, meta-reasoning process, coach) and those added during refinement. Top-level constraints can be pre-encoded or based on drives (e.g., (Coddington et al. 2005; Young & Hawes 2012)). Hard constraints in C must be satisfied at all times, while soft constraints should be satisfied if possible.

Our use of *goal memory* is distinct from its typical use in cognitive science, where goal memory is typically presented as a mental construct with representations and processes that are used to store and manage goal-related requirements of the task that a cognitive agent happened to be engaged in (e.g., Altmann & Trafton 1999; Anderson & Douglass 2001; Choi 2011). While issues such as interference level, strengthening, and priming constraints are key requirements to mimic human memory (Altmann & Trafton 2002), we ignore any such considerations because we are not concerned with the cognitive plausibility of our goal memory model.

4.2 Refinement strategies (R)

The actor applies a set of refinement strategies R to transition goal nodes in M . The Goal Lifecycle (Figure 4) captures the *possible* decision points of goals in the SDP. Decisions consist of applying a *strategy* (arcs in Figure 4) to transition a goal node N^g among *modes* (rounded boxes). For convenience, we sometimes refer to the goal node N^g as simply the goal g , though it should be clear

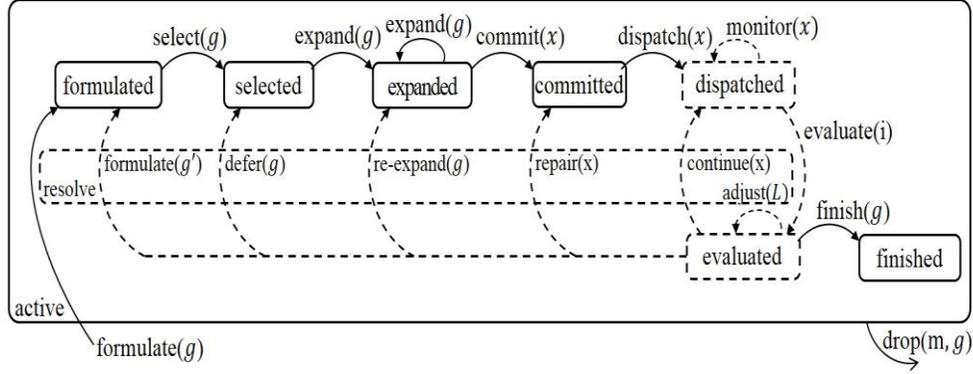


Figure 4: The Goal Lifecycle (Roberts et al. 2014). Strategies (arcs) denote possible decision points of an actor, while modes (rounded boxes) denote the status of a goal (set) in the goal memory.

that any strategies are functions that transition some N^g . In §6 we will detail strategies that we implemented for the FDR application that motivated this work.

Goal nodes in an *active* mode are those that have been formulated but not yet dropped. The **formulate** strategy determines when a new goal node is created. Vattam et al. (2013) describe goal formulation strategies. The **drop** strategy causes a goal node to be “forgotten” and can occur from any active mode; this strategy may store the node’s history for future deliberation. To **select** N^g indicates intent and requires a formulated goal node. The **expand** strategy decomposes N^g into a goal network (e.g., a tree of subgoal nodes) or creates a (possibly empty) set of expansions X . Expansion is akin to the “planning” step, but is renamed here to generalize it from specific planning approaches. The **commit** strategy chooses an expansion $x \in X$ for execution; a static strategy or domain-specific quality metrics may rank possible expansions for selection. The **dispatch** strategy slates x for execution; it may further refine x prior to execution (e.g., it may allocate resources or interleave x ’s execution with other expansions).

Goal nodes in executing modes (Figure 4, dashed lines) can be subject to transitions resulting from expected or unexpected state changes in Σ or Z . The **monitor** strategy checks progress for N^g during execution. Execution updates, including notification that the executive has completed the tasks for the goal, arrive through the **evaluate** strategy. In a nominal execution, the information can be either resolved through a **continue** strategy after which the **finish** strategy marks the goal node as *finished*.

When notable events occur during execution, the **evaluate** strategy determines how they impact goal node execution and the **resolve** strategies define the possible responses. If the evaluation does not impact N^g , the actor can simply **continue** the execution. However, if the event impacts the current execution other strategies may apply. One obvious choice is to modify the world model (i.e., Σ or Z) using **adjust**, but adjusting its model does not resolve

the mode of N^g and further refinements are required. The **repair** strategy repairs the expansion x so that it meets the new context; this is frequently called plan repair. If no repair is possible (or desired) then the **re-expand** strategy can reconsider a new plan in the revised situation for the same goal; this is frequently called replanning. The **defer** strategy postpones the goal, keeping the goal node *selected* but removing it from execution. Finally, **formulate** creates a revised goal g' ; the actor then may drop the original goal g to pursue g' or it could consider both goals in parallel.

We partition $R = R^{provided} \cup R^{added} \cup R^{learned}$ to distinguish between representations that the actor was provided prior to the start of its lifetime (e.g., through design decisions), representations that were added to its model as a result of execution in an environment (e.g., a new object is sensed), and those it learned for itself (e.g., the actor adjusts its expectations for an action after experience).

4.2 The Transition Function (δ_{GR})

Not every strategy will apply to every goal or every situation. The transition function δ_{GR} specifies the allowed transitions between modes. In a domain-independent fashion, δ_{GR} is defined by the arcs in the lifecycle. However, a system or domain may modify (through composition, ablation, or additional constraints) the transitions for M . For example, in FDR operations, human approval is required before the SDP can commit to vehicle flight paths. In such a case, additional constraints on the commit strategy would ensure that Operator consent is obtained before a vehicle actually flies a trajectory.

4.3 Instantiations of the Goal Reasoning Problem

The Goal Reasoning Problem distinguishes systems by their design choices and, thus, facilitates their comparison. Figure 5 shows how different instantiations of the Goal Lifecycle can represent iterative plan repair (e.g., Chien et al. 2000), replanning (e.g., Yoon et al. 2007), and Goal-Driven Autonomy (e.g., Klenk et al. 2013).

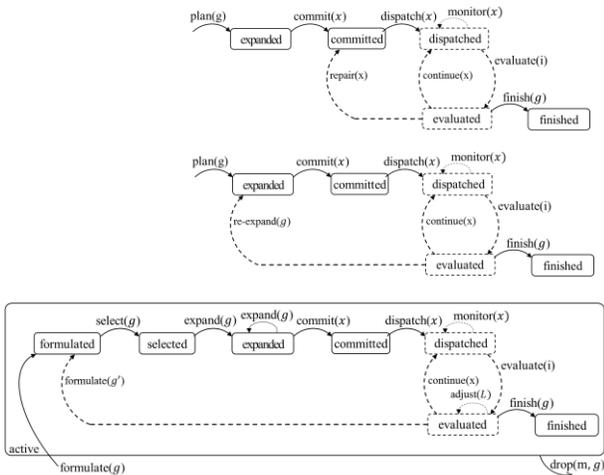


Figure 5: Instantiations of the Goal Lifecycle that incorporate plan repair (top), replanning (middle), and Goal-Driven Autonomy (bottom).

4.4 The Goal Reasoning Problem

We can now define the Goal Reasoning Problem. Let L_{init} be the initial state of the actor, which includes s_{init} . Then, the Goal Reasoning Problem can be stated:

Given Z and L_{init} , a GR actor examines its goal memory M_t at time t and chooses a strategy that maximizes its long-term rewards using $\sum_t^\infty \gamma^t \text{reward}_t$, where γ^t is a discount factor and $\text{reward}_t: M_t \times R_t \rightarrow \mathbb{R}_t$ yields the actor's reward at applying one or more refinement strategies at time t .

Roberts et al. (2014) showed how this problem could be modelled as an MDP or Reinforcement Learning problem. However, we apply neither of these in our implementation.

4.5 Related Work

The Goal Lifecycle bears close resemblance to that of Harland et al. (2014) and earlier work (Thangarajah et al., 2010). They present a Goal Lifecycle for BDI agents, provide operational semantics for their lifecycle, and demonstrate the lifecycle on a Mars rover scenario. It remains future work to more fully characterize the overlap of their lifecycle with the Goal Lifecycle we define. Work by Winikoff et al. (2010) has also linked Linear Temporal Logic to the expression of goals. Our work differs in that it focuses on teams of robots rather than single agents.

Our approach of coordinating behaviors with constraint-based planning is inspired by much of the work mentioned by Rajan, Py, and Barriero (2013) and Myers (1999). Our Team Executive leverages the Executive Assistant of Berry et al. (2003).

5. Goal Reasoning as Goal Refinement

Solutions to the Goal Reasoning Problem can be solved through refinement search, a process we call goal refinement. Goal refinement builds on planning as refinement search (Kambhampati 1994, 1997; Kambhampati et al. 1995). Refinement planning employs a split and prune model of search, where plans are drawn from a candidate space K . Let a search node N be a constraint set that implicitly represents a candidate set drawn from K . Refinement operators transform a node N_i at layer i into k children $\langle N_{j_1}, N_{j_2}, \dots, N_{j_k} \rangle$ at layer $j = i + 1$ by adding constraints that further restrict the candidate sets in the next layer. If the constraints are inconsistent then the candidate set is empty. Let N_\emptyset represent an initial node whose candidate set equals K and results from only the initial constraint set provided in the problem description (from the perspective of the search process, the refined constraints are empty, thus the subscript \emptyset). The RefinePlan algorithm recursively applies refinements to add constraints until a solution is found. A desirable property of refinements is that subsequent recursive calls result in smaller candidate subsets. Thus the constraints aid search by pruning the solution space, identifying inconsistent nodes, and providing backtracking points. Instantiations of RefinePlan correspond to variants of classical planning search algorithms. Plan refinement equates different kinds of planning algorithms in plan-space and state-space planning. Extensions incorporated other forms of planning and clarify issues in the Modal Truth Criterion (Kambhampati and Nau 1994). More recent formalisms such as angelic hierarchical plans (Marthi et al. 2008) and hierarchical goal networks (Shivashankar et al. 2013) can also be viewed as leveraging plan refinement. The focus on constraints in plan refinement allows a natural extension to the many integrated planning and scheduling systems that use constraints for temporal and resource reasoning.

Figure 6 shows a Goal Refinement algorithm. Goal Refinement begins with N_\emptyset^g , which consists of the candidate space of all possible executions achieving g . It then applies refinement strategies from the Goal Lifecycle (see Figure 4) to N_i^g at layer i into k children $\langle N_{j_1}^g, N_{j_2}^g, \dots, N_{j_m}^g \rangle$ at layer $j = i + 1$ by modifying the goal node, which further restricts the candidate sets in the next layer. Figure 7 shows how the modes of a goal indicate successively smaller candidate sets towards eventual execution; transitions between these modes consist of adding, removing, or modifying constraints and states in N^g . Each transition increases the level of commitment the actor has made to g and increases the degree of refinement for N^g . If each refinement also reduces the candidate set of solutions, then search can be more efficient.

1. RefineGoalNode (N^g, R)
2. if mode(N^g) != active
3. return
4. pick a refinement operator $r \in R$
5. refinements = apply(r, N^g)
6. if |refinements| > 1:
7. $N^{g'}$ = nondeterministically choose a refinement
8. else if |refinements| == 1:
9. $N^{g'}$ = choose refinement
10. else
11. return
12. if $N^{g'}$ is inconsistent
13. return fail
14. call RefineGoalNode($N^{g'}$)

Figure 6: A Goal Refinement algorithm

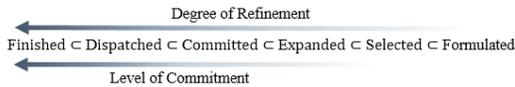


Figure 7: Modes define increasingly smaller candidate subsets for Goal Refinement

6. FDR Scenarios and GR Use Cases

Military leaders in a FDR focus on five priorities (Navy, 1996). *Relief operations* prevent or limit further loss of life or property damage; these operations focus on identifying or deploying first responders and taking actions to provide critical sustenance and first aid. *Logistics operations* establish and maintain key areas for equipment as well as plan for the distribution of materiel and personnel to the area; these focus on determining large, medium and small landing zones for air, sea, and ground vehicles as well as determining the capacity of existing infrastructure. *Security operations* locate key personnel and maintain safety for military and civilian assets; these operations involve locating the embassy and local government personnel, determining threats to operations, and providing transportation or evacuation assistance. *Communications or information sharing operations* establish and maintain an unclassified web-based network to allow foreign planners to share information with relief organizations; these involve assessing the existing communications network and possibly supplementing it as needed. *Consequence management operations* eliminate the negative impact of intentional or inadvertent release of hazardous materials as well as potential epidemics.

Two common threads in all five priorities are *updated map data* and *reliable communications*. A team of autonomous vehicles can update the map data concerning the roads and communications network, confirm the state of any potential hazardous material or threats to operations, and provide intelligence concerning the locations of survivors. We consulted with Navy reservists who perform FDR operations to develop three scenarios that showcase

how the SDP can support FDR operations. Each scenario focuses on introducing notable events or error conditions to force the SDP to respond in a coordinated way by proposing operationally relevant solutions.

Though we only discuss the scenarios in this paper, we also developed use cases based on these scenarios to independently exercise every strategy of the Goal Lifecycle. Each use case corresponds to the detection of and response to specific notable events by the Mission Manager. The use cases focus on using the **resolve** strategies of Goal Lifecycle (cf. Figure 4, dashed lines).

The vehicles in these scenarios carry three kinds of sensors. **Electro Optical (EO)** sensors that collect images. **Radio Frequency (RF)** sensors that can locate radio signals or perform radio communications. Another type of sensor detects **chemical, biological, radiation, nuclear, and explosives (CBNRE)**. We can simulate CBNRE dangers using an RF signal at a particular frequency. Alternatively, we can simulate the existence of a hazard in a mixed real-virtual environment where the vehicles are flying in the real world but sensor reports are given by a software system.

The scenarios use three vehicle types (see Figure 8). **Fixed wing Unmanned Aerial Vehicles (UAVs)** are small air vehicles such as the Bat4, Insitu ScanEagle, Unicorn or Blackjack. A UAV's operational time ranges from 2 to 20 hours, it can travel at low air speeds at altitudes up to 5000 feet, and it can carry sensor payloads up to 100kg. **Micro Aerial Vehicles (MAVs)** are small quadrotor or heptarotor UAVs such as the Acending Technologies Pelican. MAVs have operational times ranging from 3-15 *minutes*, travel close to the ground with limited range, and can carry very small EO or RF sensors. The extended scenario adds additional air and ground vehicles. **Unmanned Ground Vehicles (UGVs)** are small ground vehicles weighing about 60 pounds with a running time of 2-4 hours between charges. We use one to three iRobot Packbots PB1, PB2, and PB3. These vehicles can support significant payloads and computational power (depending on their battery life). Because MAVs are so range-limited, we also include in our scenarios **Kangaroos**, which are a combined vehicle type consisting of UGVs carrying a MAV.

These vehicles provide three atomic mission types: (1) Surveying a region with an EO sensor; (2) locate a Very Important Person (VIP) using an RF sensor; and (3) serve as a communications relay for a VIP.



Figure 8: UAV (left), UGV (middle), and MAV (right). See prose for descriptions.

6.1 Integration Scenario

This simple baseline scenario tests and demonstrates the major system components and their interactions; it involves a team of vehicles surveying the roads and finding a VIP in one of two Operator-selected regions. Three fixed-wing UAV vehicles fly over two pre-selected regions of interest to collect low resolution raster data in support of infrastructure assessment. When a UAV locates a survivor's cell phone signal, it circles the signal location acting as a relay until receiving further instruction.

Figure 9 demonstrates a hypothetical start of the integration scenario, when a known map is provided to the Coordination Layer and discretized to allow for sensor data collection. The Operator can specify that particular regions as likely to contain specific people. For example, areas around an embassy and airport are likely to have VIPs to the FDR operations. In this example, one VIP is located at a building on an embassy compound. The VIP's cell phone can be represented by any suitable radio signal emitter that the RF sensors on the vehicles will sense.

The Operator first identifies the specific vehicle/sensor platforms, which in this scenario includes three UAV vehicles, each with an EO and an RF sensor. The Operator then selects two regions and selects two missions (i.e., goals) for these regions: (1) VIPFound and (2) RoadsAssessed. Note that the Coordination Layer will eventually propose potential regions, as identified below in the extended scenario. The Coordination Layer responds by highlighting possible trajectories over these regions, soliciting operator approval, and executing the data collection for those regions. UAV1, UAV2, and UAV3 complete a survey of these two regions of interest.

UAV1 locates the VIP cell signal and responds by switching to a VIPCommsRelayed goal. The vehicle's response is to begin hovering over the VIP signal. The UI response is to add a new avatar for the VIP (e.g., a red star) in the appropriate spot on the screen. The response in the

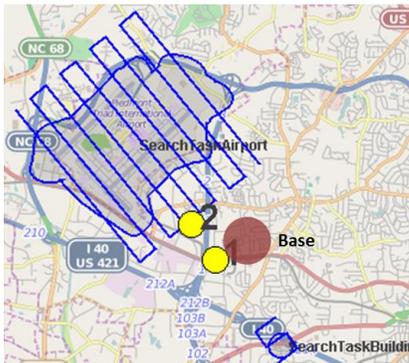


Figure 9: Example airport and VIP regions. The base is located between the regions. Also shown are the trajectories (blue lines) for two vehicles (yellow dots).

Coordination Layer will be to **formulate** a new goal VIPCommsRelayed.

The automated formulation of this goal is central to demonstrating how GR can aid the guidance of autonomous systems. One can imagine extending this to more elaborate scenarios where the system proposes new missions for the team as missions unfold.

6.2 Recommendation Scenario

This scenario extends the Integration Scenario so that the SDP additionally suggests the most likely regions for finding the VIP. In this scenario, the Operator can simply accept these regions rather than have to manually highlight them. This scenario reduces Operator entry load at the start of an FDR operation; the SDP instead refines goals to obtain the map from the world model, observe probable locations of the VIP (e.g., the embassy and the airport), and suggest regions to begin exploring. To do this, the SDP analyzes known map data to create a region around an airport, a region around a building where the VIP was last spotted, and a region following the best road network between the airport and building.

6.3 Extended Scenario

Our most ambitious scenario involves all three vehicle types and exercises every strategy in the Goal Lifecycle. Figure 10 displays the main elements of the scenario, which takes place in a region the size of a few city blocks. Streets are named along the bottom and right side of the plot; some streets have a specific region (dotted boxes near the top and middle of the plot) where vehicles will need to survey during the scenario.

This scenario extends the Integration scenario after the point where UAV1 has identified that the VIP signal is emitted near the Embassy Compound. UAV2 and UAV3 return to base, and await further instruction. For the rest of the scenario, UAV1 circles above the embassy, tracking it and acting as a communications relay. At this point, the SDP is not aware of the Flood or Chlorine Spill because trees are blocking the flood from the UAV's camera and

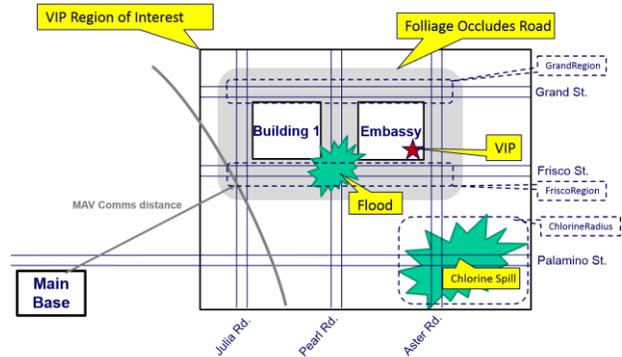


Figure 10: The Extended Scenario

the UAV is flying too high to detect the spill. The UI has already notified the operator that a VIP was found and we assume the Operator “calls” the cell phone, decides to extract this person, and adds a new goal for the autonomous team to confirm safe ingress/egress routes followed by locating the VIP within the Embassy. At the start of the scenario, the best route that can be planned to extract the VIP from the Main Base is east along Palamino St., and then north on Aster Rd., which we denote as Route A. The Coordination Layer suggests allocating two Kangaroos to explore Route A, confirms this route via the UI Layer with the human, and dispatches the approved trajectories.

During navigation of Route A, the Kangaroos detect a chlorine spill of unknown severity. The Mission Manager responds by suggesting a new allocation of three additional MAVs with CBNRE sensors to assess the extent of the spill. Alternatively, it could assign one Kangaroo to stop exploring Route A and help with this task. By now the Kangaroos should be at or near the VIP. However, because of the chlorine spill, a new route must be determined. The Mission Manager asks the Kangaroos to survey the Frisco Region (dashed lines on Frisco St.) and the flood is discovered, so Frisco St. is deemed impassable until further inquiry is done (this follow up goal should appear in the goal list of the Coordination Layer). The Mission Manager then commands the Kangaroos to survey the GrandRegion and determine Grand St. is passable. A safe route between the base and VIP is now established. The Kangaroos enter the embassy and locate the VIP. Because the FDR mission context includes locating survivors in buildings, the Mission Manager suggests the Kangaroos perform additional searching within the embassy to the Operator, who consents to this goal.

7. Goal Refinement for FDR

In this section, we detail our implementation of Goal Refinement in the SDP. We encoded the domain knowledge as a hierarchical goal network (Shivashankar et al. 2013; Geier & Bercher 2012). The SDP’s goal network is currently hand-coded, but we are currently writing this model in the ANML language (Dvorák et al. 2014) and plan to integrate a full planning system in the SDP. The SDP will eventually guide vehicles in cooperation with its Operator, but in this paper we assume static mission goals and a fixed number of vehicles. We implemented the Goal Refinement model as a Java library and used this library to implement a goal network for the integration scenario. In this section, we provide details regarding how we implemented the GR actor for the Mission Manager of the SDP. We will frequently refer the goal nodes as “goals” to

simplify the explanation, though it should be clear from context that these are actually the goal nodes of §4.1.

A GoalMemory class stores goal nodes in a priority queue sorted by the node priority. The priority of a node in the SDP depends on mode: formulated (10000), selected (20000), expanded (30000), committed (40000), dispatched (50000), evaluated (70000), and finished (90000). Higher priority ensures that goals further along in the lifecycle get more attention.

The Mission Manager is the GR actor in the SDP. It works with two GoalMemory objects. An InternalGoalMemory is used to initialize the system (e.g., load the domain) and store incoming information waiting to be processed. It stores any goals drawn from $L_{internal}$ (see §4). A DomainGoalMemory stores the goal nodes associated with $L_{external}$. Figure 11 displays the goal network stored in the DomainGoalMemory for the scenario in Figure 9.

Goals (i.e., goal nodes) in the SDP exist within a goal ontology. Strategies in the SDP are written as a Strategy Pattern (Gamma et al. 1994) called by the Mission Manager. Every goal in the SDP implements the StrategyInterface; goals override methods in this interface to implement their specific strategies. Unless specifically stated as overridden, goals inherit the strategies of the goals they extend. This allows default strategies to be implemented high in the goal ontology, encouraging strategy reuse.

StrategyInterface contains the following strategies: **formulate**, **select**, **expand**, **commit**, **dispatch**, **evaluate**, **resolve**, **finish**, and **drop**. This interface provides no default implementation for these strategies.

BaseGoal (implements StrategyInterface) and defines the default strategies for all goals. It also implements the δ_{GR} transition function that controls which strategies can be applied; all strategies consult δ_{GR} to ensure a transition is allowed. The **formulate** strategy creates the appropriate goal node and insert the node in the correct GoalMemory. The default behavior for most strategies is to transition to the next mode if the strategy is called. However, the **expand** and **finish** strategies perform additional steps. The **expand** strategy for a goal may require interaction with a process that provides the expansions (e.g., it may require a planner). So it has a hook that allows subclasses to specify how expansions can be generated if needed. If no expansion process is provided for a goal subclass, the default behavior is to allow expand to continue. The default **finish** strategy confirms that all subgoals are finished before allowing a parent to transition to *finished*.

Goal Description	Goal Type
⚙️ AACS Domain Root	OperationalGoal
⚙️ Logistics Operations	OperationalGoal
⚙️ Assess Infrastructure	OperationalGoal
⚙️ Assess Airport LZ	AchieveTeamMission
Mission:Assess Airport LZ	VehicleMission
⚙️ Security Operations	OperationalGoal
⚙️ Maintain VIP safety	OperationalGoal
⚙️ Assess VIP Region(s)	AchieveTeamMission
Mission:Assess VIP Region(s)	VehicleMission
⚙️ Relay VIP if found	AchieveTeamMission
Mission:Relay VIP if found	VehicleMission

Figure 11: Goal decomposition during an SDP run

When the Mission Manager first loads, it places goals in each memory and adds a goal to load the domain.

MaintainGoalMemory (extends BaseGoal) is the goal that ensures goals continue to get processed in each GoalMemory. This goal contains a queue of modified goals and notifies the MissionManager of changes. When a goal is modified, the MaintainGoalMemory goal tries to apply the next applicable strategy.

AchieveDomainLoaded (extends BaseGoal) is a goal that fills the root goal in the DomainGoalMemory. It currently places the root goal in the DomainGoalMemory and finishes. However, if there were database calls or files to read for the domain to load, the **expand** strategy would be the correct strategy to implement this functionality.

Non-primitive domain goals are expanded by instantiating a sub-goal tree for the goal. The SDP commits to and dispatches the only expansion available for these goals, since this is a small example. These non-primitive goals remain in a dispatched (i.e., in-progress) state until their subgoals finish. The remaining goals are domain specific and relate to Figure 11.

OperationalGoal (extends BaseGoal) is base type for non-primitive goals that match to the operational priorities of FDR operations. Strategies for these goals are the same as the BaseGoal.

Operational subgoals eventually decompose into AchieveTeamMission goals.

AchieveTeamMission (extends BaseGoal) modifies two strategies from the default given by BaseGoal. The **expand** strategy connects a special trajectory generator to create expansions. The trajectory generator examines many factors to create a suitable trajectory. However, the details of this are not known to the goal. The trajectory generator implements an interface that returns an expansion that is attached to the goal node after calling

expand. The **commit** strategy requires Operator approval before it can send vehicles on a trajectory. So this strategy confirms that approval has been granted.

Expanding an AchieveTeamMission goal results in a specific VehicleMission goal, which includes details regarding a proposed allocation of a vehicle to a specific trajectory (cf. the lawnmower flight paths of Figures 1 and 9). Once the VehicleMission details are approved – either automatically or by the Operator – the Mission Manager commits and dispatches the proposed expansion of the VehicleMission for execution.

VehicleMission (extends BaseGoal) is a goal that allows the Mission Manager to track the progress toward completion of a mission. It modifies only one strategy from the default behavior. The **dispatch** strategy sets up a special listener for vehicle state that triggers the goal to move to call the evaluate strategy when a vehicle update is received. Because the default behavior of every strategy is to attempt to move to the next mode, the **evaluate** strategy will move to finished when the progress of a vehicle is above a specific threshold that indicates the task is complete.

The Coordination Manager and Team Executive then begin sending vehicle commands. The VehicleMission goal remains dispatched until new information (e.g., a progress update) causes it to become finished or need some other resolve strategy.

The Mission Manager has triggers to monitor the dispatched goals so that it will notice if the goal is stalled or completed by the executive. The Mission Manager uses a repair strategy on the original vehicle allocation to retask a vehicle for a stalled VehicleMission,

8. Demonstration

We demonstrate the SDP on the Integration Scenario (see §6.1). Figure 9 shows an airport region (upper left) and, 3-5 km away from the airport, a VIP region (lower middle) that is centered on a particular building near the suspected location of the VIP. The VIP emits a radio signal (e.g., cell phone signal). Two fixed-wing air vehicles (in yellow) are tasked with assessing the two regions and finding the VIP. They carry Electro Optical and Radio Frequency sensors that activate when the target is within their sensor radius.

The integration scenario demonstrates the SDP's key capabilities, namely that: (1) the SDP can create new goals responding to an open world (e.g., it collectively responds to the VIP being found); (2) a vehicle can make decisions autonomously (e.g., a vehicle may begin relaying the VIP once found); (3) the SDP responds to vehicles making

autonomous decisions (e.g., it notes the vehicle relaying instead of surveying when the VIP is found); and (4) the SDP can retask a vehicle to complete a mission (e.g., it retasks stalled missions to idle vehicles).

To generate 30 scenarios based on Figure 1 we select 30 random airports from OpenStreetMaps data for North Carolina (Geofabrik 2014) and then select buildings within 3-5 kilometers of the airport. Buffer regions of 300 meters around the airport and the building serve as the airport and VIP regions, respectively. Each run completes when (1) both regions are completely surveyed and the VIP is found or (2) the simulation reaches 35,000 steps. Each step is approximately one second of real time simulation. We use the MASON simulator (Luke et al. 2005) to run the scenario.

At the start of the scenario, one vehicle is assigned to assess the Airport Region, denoted by AirportVehicle, and the other vehicle is assigned to the VIP Region, denoted VIP Vehicle. Vehicles return to the base when their fuel is sufficiently low. Vehicle behavior depends on whether the vehicles can retask themselves to relay when the VIP is found (denoted *+Relay*) or they do not relay (*-Relay*). Regardless of whether a vehicle begins relaying, the Mission Manager should always create a new “Relay VIP” goal when the VIP is found. The Mission Manager behavior depends on whether it is allowed to retask a vehicle (*+Retask*) or not (*-Retask*).

Condition 1: Find VIP (*-Relay -Retask*) provides a baseline. In it the vehicles detect the VIP and a new goal to relay the VIP appears when the VIP is found. Getting the SDP to do something meaningful with the “Relay VIP” goal is our next condition.

Condition 2: Relay VIP (*+Relay -Retask*) demonstrates that a vehicle can retask itself with a new goal by automatically relaying the VIP once found. The retasking is embedded in the Vehicle Controller (see Figure 5, line 15). However, this change of behaviors needs to be shown in the goal network, where the goal “Mission: RelayVIP” should appear after the VIP is found. However, nothing is done with the new goal and VIP Vehicle does not complete the entire survey of the VIP region because it switches its own task to relaying.

Condition 3: Relay and Retask (*+Relay +Retask*). To address the problem of the VIP region remaining unfinished, the Coordination Layer is allowed to retask the Airport Vehicle so it finishes the VIP Region survey after completing its area first.

When we run the simulation on the three conditions, we observe exactly the expected results. In every case, a new goal is observed in the Mission Manager after the VIP is found. In the Relay VIP condition, the VIP Vehicle begins relaying as expected, leaving the VIP Region unfinished. When the Mission Manager is allowed to retask vehicles, we observe that all three missions complete.

This demonstration exercises most Goal Lifecycle strategies (i.e., all except **adjust** and **re-expand**). It should be clear that the nominal strategies (cf. Figure 4, solid arcs) are executed. However, it may be less clear that the demonstration also applies most of the **resolve** strategies (Figure 4, dashed lines). During the notable events of the scenario, the **evaluate** step notifies the Mission Manager of the need to deliberate. When a vehicle returns to base to recharge, the Mission Manager applies **continue** because recharging is an expected contingency behavior of surveying. When relaying is enabled (*+Relay*) and a vehicle switches to relaying, the Mission Manager must apply **formulate(g')** where g' is the RelayVIP goal. Then it applies **defer(Survey)** in preference to the RelayVIP goal. This leaves the Survey goal in a selected (i.e., unfinished) mode. With retasking enabled (*+Retask*), the Mission Manager applies **repair(Survey)** to reassign the goal to the AirportVehicle. Running the ablated versions (i.e., *-Relay* or *-Retask*) is the same thing as limiting the strategies available to the Mission Manager.

9. Summary and Future Work

We detailed our implementation of a prototype system, called the Situated Decision Process (SDP), which uses a Goal Refinement library we have constructed to coordinate teams of vehicles running LTL-synthesized FSAs in their vehicle controllers. The central contributions of this paper are clarifying the relationship of GR with Nau’s (2007) model of online planning, more clearly defining the Goal Reasoning Problem and Goal Refinement, outlining a set of use cases for Foreign Disaster Relief (FDR), detailing the implementation of Goal Refinement for FDR in our prototype system and demonstrating that the SDP can respond to notable events during execution.

Future work will consist of further automating portions of the SDP, extending the demonstration to work with large teams of robotic vehicles, and enriching the domain model. For example, we plan to extend the domain model to fully encode temporal and resource concerns similar to the TREX system (Rajan, Py, and Barriero 2013). We plan to test the SDP against the more challenging scenarios with richer sensor models and higher-fidelity simulations. Ultimately, we plan to run the SDP on actual vehicles and perform user studies on its effectiveness in helping an Operator coordinate a team of vehicles in Disaster Relief.

Acknowledgements

Thanks to OSD ASD (R&E) for sponsoring this research. The views and opinions in this paper are those of the authors and should not be interpreted as representing the views or policies, expressed or implied, of NRL or OSD. We also thank the reviewers for their helpful comments.

References

- Anderson, J. R., & Douglass, S. (2001). Tower of Hanoi: Evidence for the cost of goal retrieval. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 27, 1331–1346.
- Altmann, E. M., & Trafton, J. G. (1999, August). Memory for goals: An architectural perspective. In *Proc. of the 21st annual meeting of the Cognitive Science Society* (Vol. 19, p. 24).
- Apker, T., Liu, S.-Y., Sofge, D., and Hedrick, J.K. (2014). Application of grazing-inspired guidance laws to autonomous information gathering. *Proc. of the Int'l Conference on Intelligent Robots and Systems* (pp. 3828-3833). Chicago, IL: IEEE Press.
- Balch, T., Dellaert, F., Feldman, A., Guillory, A., Isbell, C.L., Khan, Z., Pratt, S.C., Stein, A.N., & Wilde, H. (2006). How multirobot systems research will accelerate our understanding of social animal behavior. *Proc. of the IEEE*, 94(7), 1445-1463.
- Berry, P., Lee, T. J., & Wilkins, D. E. (2003). Interactive execution monitoring of agent teams. *Journal of Artificial Intelligence Research*, 18, 217–261.
- Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., Sa'ar, Y. (2012). Synthesis of Reactive(1) designs. *Journal of Computer and System Sciences*, 78(3), 911–938.
- Chien S., Knight R., Stechert A., Sherwood R., and Rabideau, G. (2000) Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling. *Proc. of the Conf. on Auto. Plan. and Sched.*(pp. 300-307). Menlo Park, CA: AAAI.
- Choi, D. (2011). Reactive goal management in a cognitive architecture. *Cognitive Systems Research*, 12(3), 293-308.
- Coddington, A.M., Fox, M., Gough, J., Long, D., & Serina, I. (2005). MADbot: A motivated and goal directed robot. *Proc. of the 20th Nat'l Conf. on Art. Intel.*(pp. 1680-1681). Pittsburgh, PA: AAAI Press.
- Dvorák, F., Bit-Monnot, A., Ingrand, F., & Ghallab, M. (2014). A Flexible ANML Actor and Planner in Robotics. In *Working Notes, PlanRob Workshop at ICAPS*. Portsmouth, NH: AAAI.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.
- Geofabrik. *OpenStreetMap Data Extracts*. (2014) Accessed from <http://download.geofabrik.de/index.html>.
- Ghallab, M., Nau, D., & Traverso, P. (2014). The actor's view of automated planning and acting: A position paper. *Artificial Intelligence*, 208, 1–17.
- Geier, T. & Bercher, P. (2011). On the decidability of HTN Planning with task insertion. In *Proc. of the 22nd Int'l Joint Conf. on AI*. (pp. 1955-1961). Barcelona: AAAI.
- Harland, J., Morley, D., Thangarajah, J., & Yorke-Smith, N. (2014). An operational semantics for the goal life-cycle in BDI agents. *Auton. Agents and Multi-Agent Systems*, 28(4), 682–719.
- Ingrand, F., & Ghallab, M. (2014). Robotics and artificial intelligence: A perspective on deliberation functions. *AI Communications*, 27(1), 63-80.
- Kambhampati, S., Knoblock, C.A., & Yang, Q. (1995). Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Art. Intelligence*, 76, 168-238.
- Kambhampati, S. & Nau, D. (1994). On the nature of modal truth criteria in planning. *Proc. of the 12th Nat'l Conference on AI* (pp. 67-97). Seattle, WA: AAAI Press.
- Klenk, M., Molineaux, M., & Aha, D.W. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Comp. Intell.*, 29(2), 187-206.
- Kress-Gazit, H., Fainekos, G.E., & Pappas, G.J. (2009). Temporal logic based reactive mission and motion planning. *Transactions on Robotics*, 25(6), 1370-1831.
- Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., & Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81(7), 517-527.
- Marthi, B, Russell, S., & Wolfe, J. (2008). Angelic hierarchical planning: Optimal and online algorithms. *Proc. of the Int'l Conf. on Auto. Plan. & Sched.* (pp. 222-231). Menlo Park, CA: AAAI.
- Myers, K.L. (1999). CPEF: A continuous planning and execution framework. *AI Magazine*, 20(4), 63-69.
- Nau, D. (2007) Current trends in Automated Planning. *AI Magazine*, 28(4), 43-58.
- Navy, U.S. Department of. (1996) Humanitarian assistance/disaster relief operations planning, (TACMEMO 3-07.6-05). Washington, D.C.: Gov't Printing Office.
- Pollack, M.E., & Horty, J. (1999). There's more to life than making plans: Plan management in dynamic, multiagent environments. *AI Magazine*, 20, 71-83.
- Rajan, K., Py, F., & Barreiro, J. (2012). Towards deliberative control in marine robotics. In *Marine Robot Autonomy* (pp. 91–175). New York, NY: Springer.
- Roberts, M., Vattam, S., Alford, R., Auslander, B., Karneeb, J., Molineaux, M., Apker, T., Wilson, M., McMahon, J., & Aha, D.W. (2014). Iterative goal refinement for robotics. In *Working Notes of the Planning and Robotics Workshop at ICAPS*. Portsmouth, NH: AAAI.
- Roberts, M., Apker, T., Johnson, B., Auslander, B., Wellman, B. & Aha, D.W. (2015). Coordinating Robots for Disaster Relief. *Proc. of the Conf. of the Florida AI Research Society* (to appear) Hollywood, FL: AAAI.
- Shivashankar, V., Alford, R., Kuter, U., & Nau, D. (2013). The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. *Proc. of the 23rd Int'l Joint Conference on AI* (pp. 2380-2386). Beijing, China: AAAI.
- Smith, D., Frank, J., & Jonsson, A. (2000). Bridging the gap between planning and scheduling. *Know. Eng. Rev.*, 15, 61-94.
- Thangarajah, J., Harland, J., Morley, D., & Yorke-Smith, N. (2011). Operational behaviour for executing, suspending, and aborting goals in BDI agent systems. In *Declarative Agent Lang. and Technologies VIII* (pp. 1–21). Toronto, Canada: Springer.
- Vattam, S., Klenk, M., Molineaux, M., & Aha, D. W. (2013). Breadth of approaches to goal reasoning: A research survey. In D.W. Aha, M.T. Cox, & H. Muñoz-Avila (Eds.) *Goal Reasoning: Papers from the ACS Workshop* (Tech. Report CS-TR-5029). College Park, MD: Univ. of Maryland, Dept. of Comp. Science.
- Winikoff, M., Dastani, M., & van Riemsdijk, M. B. (2010). A unified interaction-aware goal framework. In *Proc. of ECAI* (pp. 1033–1034). Lisbon, Portugal: IOS Press.
- Yoon, S.W., Fern, A., & Givan, R. (2007). FF-Replan: A baseline for probabilistic planning. *Proc. of the 17th Int'l Conf. on Auto. Plan. and Sched.* (pp. 352-359). Providence, RI: AAAI Press.
- Young, J., & Hawes, N. (2012). Evolutionary learning of goal priorities in a real-time strategy game. In *Proc. of the 8th AAAI Conf. on Artificial Intelligence and Interactive Digital Entertainment*. Stanford, CA: AAAI Press.