Ethical Mission Definition and Execution for Maritime Unmanned Systems: A Practical Approach

Don Brutzman, *Member IEEE*, Duane Davis, *Member IEEE*, Curtis Blais, *Member, IEEE*, and Robert B. McGhee, *Fellow, IEEE*

***Abstract*— Many experts and practitioners have worked long and hard towards achieving functionally capable robots. While numerous areas of progress have been achieved, progress in ethical control of unmanned systems has been elusive and problematic. Common conclusions that treat ethical robots as an always-amoral philosophical conundrum or requiring undemonstrated morality-based artificial intelligence (AI) are simply not sensible or repeatable. For better or worse, actors around the world are rapidly designing and deploying mobile unmanned systems to augment human capabilities. Thus theory must meet practice. This work adapts policies and procedures for ethical responsibility and authority that have been proven to work in collaborative military operations, even across varying cultures and platforms. Patterning after successful practice by human teams shows that precise mission definition and task execution can provide safeguards for autonomous robots or human-robot teams possessing potentially lethal capabilities. Since lethality is not limited to military weapons but can also include navigational interference and vehicle collisions, and since many robots are capable of carrying out well-defined tasks regardless of their internal software architecture, this approach appears to have broad usefulness for civil application of unmanned systems as well.**

***Index Terms*—autonomous vehicles, ontology, robotics, robot ethics**

I. Nature of Ethical Missions

MANY experts and practitioners have worked long and hard towards achieving functionally capable robots. While numerous areas of progress have been achieved, progress in ethical control of unmanned systems has been elusive and problematic. Common conclusions that treat ethical robots as an always-amoral philosophical conundrum or requiring undemonstrated morality-based artificial intelligence (AI) are simply not sensible or repeatable. For better or worse, actors around the world are rapidly designing and deploying mobile unmanned systems to augment human capabilities. Thus theory must meet practice.

This work adapts policies and procedures for ethical responsibility and authority that have been proven to work in collaborative military operations, even across varying cultures and platforms. Patterning after successful practice by human teams shows that precise mission definition and task execution can provide safeguards for autonomous robots or human-robot teams possessing potentially lethal capabilities. Since lethality is not limited to military weapons but can also include navigational interference and vehicle collisions, and since many robots are capable of carrying out well-defined tasks regardless of their internal software architecture, this approach appears to have broad usefulness for civil application of unmanned systems as well.

Experience and experimentation across four decades of robotic and military operations inform this work. The authors first look at unmanned capabilities and limitations, along with real-world exemplars of how humans delegate command responsibility and authority. Robot mission tasks and goals can be clearly specified and refined with corresponding degrees of internal control supervision occurring, in the case of the exemplar discussed here as part of a three-layer software architecture. The Autonomous Vehicle Command Language (AVCL) (Davis, D. T. 2006) (Brutzman, Davis, et al. 2013) (Brutzman et al. 2013) allows expressing such mission constructs in a formal yet human-understandable way, matching the repertoires of most human-driven and robot-supervised vehicles. Adding well-defined prerequisite constraints (permission, restriction, and required human intervention) can supplement mission orders in context of each individual task, providing an ethical basis for unmanned system tasking that matches human understanding of similar responsibilities. Careful structuring of Mission Execution Automata (MEA) demonstrates a theoretically sound and scalable basis to this approach. The functional vocabulary is intentionally restricted to the well-understood mission capabilities of humans and robots so that broad compatibility by many robots is possible. Strict-subset vocabularies might alternatively implement these atomic concepts using slightly different syntax, but the core concepts must remain consistent.

Modeling, simulation and visualization have enabled extensive testing of mission operations, building human confidence in well-defined task orders. XML validation of AVCL tasks confirms syntactical correctness of mission orders, but more is needed. The authors therefore have created a

Paper submitted for review on 29 January 2017.

This work was supported in part by the Naval Postgraduate School (NPS) Consortium for Robotics and Unmanned System Education and Research (CRUSER) and the Office of the Secretary of Defense (OSD) Joint Ground Robotics Enterprise (JGRE).

Don Brutzman (brutzman@nps.edu), Duane Davis (dtdavis@nps.edu), Curtis Blais (clblais@nps.edu), and Robert McGhee (e-mail: robertbmcghee@gmail.com) are faculty at Naval Postgraduate School (NPS), Monterey California USA 93943.

Mission Execution Ontology based on principles of description logics, and implemented using Semantic Web languages. This ontology is used to confirm that mission definitions are also semantically complete, including ethical constraints whenever appropriate. Such pre-mission verification of mission completeness is analogous to chain-of-command human review of operations orders that already occurs prior to coordinated team operations.

A long trail has led to this point, inspired by many sources but driven by a need to implement practical constraints on unmanned systems lethality. A feasible path forward now exists. Semantic coherence of mission orders for humans and robots working together can be achieved, if tasks include ethical constraints that define acceptable operational prerequisites for remote action. Current project conclusions show that much work remains for ethical control of robots, but progress is indeed possible and quite encouraging. The authors believe that ethical human supervision of semiautonomous unmanned systems is feasible today and widely repeatable in a practical manner.

II. Considering Critical Challenges

The idea of intelligent robots emerged from and developed in the minds of artists and dreamers long before the prevailing technology was capable of supporting its underlying premises. First imagined using the term "robot" in the Czech play "*Rossum's Universal Robots*" (Čapek 1921), these intelligent humanoid machines were relegated primarily to the realm of science fiction in the first half of the twentieth century. Even so, the ethical ramifications of mobile (and potentially lethal) machines capable of human-like intelligence and actions were readily apparent, and seemingly reasonable ethical frameworks, most notably Azimov's Three Laws of Robotics (Azimov 1950), were devised to govern intelligent robot operation. As science fiction aficionados are well aware, however, these frameworks were rife with loopholes and unanticipated subtleties that inevitably led to their downfall.

The advent of digital computing, the emergence of artificial intelligence as an academic discipline, and the simultaneous incorporation of both into a variety of robotic devices have brought these ethical concerns to the forefront of academic and practical debate. Moreover, the ready availability of this technology to governments, corporations, research entities, and individuals has made this issue one of broad societal importance. From robotic vacuum cleaners to armed military drones, intelligent robotic technology has insinuated itself into aspects of our lives that were not previously imagined. One implication of this ubiquity is that questions of legal and moral responsibility will not be answered by a set of fixed "laws" and cannot be regulated into irrelevance through government action, just as is the case of endeavors involving humans.

Nevertheless, a number of important observations can be made:

- *Predictability*. Robots essentially perform exactly as programmed to perform in a given situation. Predictability is independent of the intent of the programmer, the understanding of the operator, and any anthropomorphic bias of observers. Fault-detection logic can detect unexpected errors and prevent unplanned actions. Robots do precisely what the programmers and operators tell them to do—not what the programmers and operators thought they told them to do or meant to tell them to do. Thus a trustworthy robot must be competent to perform assigned tasks.
- *Authority*. Apparent intelligence notwithstanding, a robot is an inanimate object. Thus, moral responsibility for the consequences of a robot's actions cannot be assigned to the robot. Thus decision-making authority must be performed by qualified, well-informed humans.
- *Responsibility*. Direct responsibility for the outcomes of robot activity must accompany authority, and must be assignable to a specific human entity. For robot ethics to bear any tangible meaning, ultimate moral and legal accountability must reside with the human programmers, manufacturers, operators and leadership. Deliberate care must be taken when giving orders to robots, just as is already given for orders to humans.
- *Liability*. The assignment of liability (whether legal or moral) in any circumstance is premised on the assumption that the involved parties are in a position to reasonably foresee the outcomes for which they are being held responsible. Liability accompanies authority and responsibility. If a human decides to deploy an unmanned system capable of lethal action, the human is liable for its subsequent actions (even if unforeseen).

These observations are fairly widely accepted, but nevertheless can lead ethicists to different conclusions. In debating military use of autonomous systems, for instance, Rob Sparrow of the International Committee for Robot Arms Control uses *Jus en Bello* requirements to argue that the military use of lethal robots is inherently unethical because robots cannot be held accountable for their actions (Sparrow 2012). Ronald Arkin, on the other hand, accepts the premises of Sparrow's argument but comes to the opposite conclusion—that if an autonomous system is capable of making a lethal decision more reliably than a human, then it is inherently unethical to *not* use that system (Arkin 2009).

Notwithstanding disagreements over military use of autonomous robots, these observations can form a common basis that provides a framework for ethical operation of intelligent robots. This approach is feasible with current technologies and without a requirement for black-box artificial intelligence "ethical controllers" that do not integrate well with specialized software schemes and inevitably lead to second-guessing, obfuscation, and uncertainty. Further, this paradigm is potentially applicable not only to military operations (lethal or otherwise) but also to other employment of robotic systems where questions of ethical operation and responsibility arise. (Lin, Abney and Bekey 2011) (Scharre 2016)

III. Military Operations as an Analogy

A. Formal Military Guidance Continues to Evolve

Updating military guidance on use of autonomous systems has become a perennial exercise in producing future roadmaps

and plans. Such documents are becoming more mature and consistent, so current guidance (although plentiful) can provide helpful insights.

Department of Defense Directive 3000.09, Autonomy in Weapon Systems (Department of Defense 2012) identifies key principles (policy) to be met in "the development and use of autonomous and semi-autonomous functions in weapon systems, including manned and unmanned platforms" and "establishes guidelines designed to minimize the probability and consequences of failures in autonomous and semi-autonomous weapon systems that could lead to unintended engagements":

"Autonomous and semi-autonomous weapon systems shall be designed to allow commanders and operators to exercise appropriate levels of human judgment over the use of force. … measures will ensure that autonomous and semi-autonomous weapon systems: (a) Function as anticipated in realistic operational environments against adaptive adversaries.(b) Complete engagements in a timeframe consistent with commander and operator intentions and, if unable to do so, terminate engagements or seek additional human operator input before continuing the engagement." (Department of Defense 2012, 2)

The Directive speaks of such systems as "human-supervised autonomous weapon systems." As such the systems need to fall under the same rules and constraints as their manned counterparts. We expect future guidance to expand these principles to human-robot teams, despite the lack of commonality that is implemented in robotic systems

B. Command Responsibility in Military Operations

Issues of authority and responsibility hold great importance in military operations, providing many useful analogies with essential relevance that are well understood. Military commanders are provided forces over which they exercise control and assigned missions that they are expected to accomplish. Responsibility for the success, failure, and conduct of the mission rests solely with the commander. This paradigm is applied at all levels of command, from the individual soldier responsible only for his own conduct to the overall commander responsible for the entire operation. From a practical standpoint, this means commanders are responsible for the proper employment of all assigned assets, whether they are in positions to actively direct those assets' conduct or not. Naval leaders in particular are frequently required to assume responsibility over units with which they have little or no direct contact. More recently, militaries have relied on increasingly automated systems. The increased capability of these systems, however, does not obviate the commander's responsibility for their proper employment. Ultimately, it does not matter whether a military leader is employing a system of people or a system of machines: authority requires responsibility. (Mack, Seymour and McComas 1998)

As stated previously, the ability to assign responsibility for operational decisions is premised on the ability of the decision-maker to reasonably foresee the results. In military operations, this requires a level of trust on the part the commander that is based on a number of important factors. Each is directly matched by similar protocols for humans.

a. Qualification. The subordinate unit must be trained and qualified for the designated task requirements, or else the acting agent (human or robotic) is simply not prepared and ready for the desired assignment.

b. Comprehension and acknowledgement. The subordinate unit must understand both tasking and on-board capabilities to conduct mission tasks according to the commander's direction. Acknowledging (or responding negatively to) the provided tasking is necessary for the commander to know that orders have been received and can be carried out.

c. Recognizes status and completion. Finally, any subordinate unit or employed system must be able to accurately assess the state of the task's execution over the course of the assignment, and can determine the ongoing status of any constraints that have been imposed on the execution. Success or failure must be recognizable.

Essentially, these trust relationships provide assurance that a properly employed system (human or otherwise) has the ability to operate in a manner that does not impose undue risk. A system that is insufficiently capable or improperly employed system, on the other hand, will not meet this operational standard, and a human commander will rightfully be held responsible for any unwanted outcomes.

Launching indiscriminate weapons is not a lawful act. These moral and legal principles are well understood by military professionals. Robots tasked with serious missions must be capable of carrying them out safely and in an ethical (professional) manner, just like any other qualified wingman or shipmate.

C. Applicability to Military and Civilian Operation of Robotic Systems

A variation of this ethical mechanism upon which military command accountability rests can be applied to robots in both military and civilian applications in a fairly straightforward manner as a corollary to the well-established legal principle of *vicarious liability* (Vicarious Liability 2011). Under this mechanism, operators can be held morally and legally responsible for all outcomes of a robot system's operation if they are in a position to foresee those outcomes. That is, operators can be held responsible for undesirable outcomes that they are in a position to prevent. Such outcomes may be highly significant, from both moral and legal perspectives, if property or lives are lost.

All robots are designed with a set of basic operational and sensory capabilities, the complexity of which varies from robot to robot. A simple drone, for instance, might be capable of no more than accurate transit between geographic waypoints, while a more-capable vehicle might be able to derive and execute a complex coverage pattern to search an area. Similarly, many robots are capable of GPS navigation, but more-advanced robots might also be capable of determining their position relative to specific geographic features.

D. Trust in Capabilities is Necessary

It is reasonable under this understanding of robot actions to assume a level of trust in a particular robot's ability to satisfactorily execute each of their atomic capabilities. It follows then, that it is possible for the robot operator to task that robot with enough confidence to assume moral and legal liability for its conduct (Lokhorst and ven den Hoven 2012). If such trust is misplaced, then the manufacturer or programmer is presumably "design responsible" for their flawed product (Lokhorst and ven den Hoven 2012). Thus responsibility and culpability can be determined throughout this long line of human involvement with any given unmanned system.

From a practical standpoint, the efficacy of this proposed framework rests on three requirements that in combination can provide robot operators with the required level of mission assurance and understanding to assume responsibility for the robot's conduct:

- **Semantic correctness**. The operator must be able to provide a rigorous mission definition that he himself fully understands. This understanding must be mathematically sound in the sense that the operator must be able to assure himself that the mission will progress from one atomic task to the next as the operator intends under all foreseeable circumstances.
- **Consistent clarity**. At its lowest level, the mission definition and any constraints must be defined in a form that is acceptable to the assigned vehicle itself. That is, it is not sufficient that the operator understand a version of the mission that is eventually translated into actual vehicle commands. Stated differently, there must be no means by which the operator-approved mission description can be semantically modified between the time that the mission is approved and the time it is executed by the target vehicle.
- **Executable composition**. The mission tasking and associated constraints must be comprised entirely of atomic vehicle-specific behaviors for which the target vehicle has been afforded an acceptable level of trust. The vehicle must also be trusted to execute the individual mission tasks, similarly trusted as able to recognize when a proposed (or active) task cannot be successfully executed. The unmanned system must be able to continually evaluate all execution constraints and determine when one is likely to be violated.

Upcoming sections of this paper present a formalization of the way that some missions (such as operations orders) can be developed and defined for human execution. In particular, a specific mission is described in structured natural language, and then, after a series of steps, abstracted into a ternary-branching "process flow diagram" that includes run-time ethical constraints. Subsequently, in a following section, it is shown how such a graph can be reinterpreted as mission orders for joint human/robot collaboration or entirely autonomous execution.

IV. Mission Definition as Goals or Tasks with Run-Time Constraints

A. Goal Definition and Task Decomposition

In describing complex tasks to subordinates, humans often subdivide these tasks into a series of subordinate tasks that can be executed in order to accomplish the overall mission. For instance, a complex task (or mission) during which a manned vehicle is expected to conduct searches and collect environmental samples before rendezvousing with another manned (or unmanned) vehicle might be specified as a series of tasks as depicted in Figure 1. Providing the vehicle's operator knows the geographic characteristics of Areas A, B, and C and understands what the commander means in directing searches, environmental sampling, and rendezvous, the operator is able to reliably execute this mission as specified.

Task 1:	Proceed to Area A and search the area.
Task 2:	Obtain an environmental sample from Area A.
Task 3:	Proceed to Area B and search the area.
Task 4:	Proceed to Area C and rendezvous with vehicle 2.
Task 5:	Proceed to recovery position (mission

Figure 4. Example mission orders expressed in structured natural language for human execution.

Note that each of the above tasks is nontrivial. Most tasks include transit as well as subsequent operations in different locations. Each task requires multiple sophisticated steps for successful completion, whether accomplished by a human or a robot. Each subtask typically requires even more specialized capabilities. For example transit requires safe navigation, which requires sensing and classification for situational awareness plus stable control, which in turn requires operation of hardware/software capabilities, and so on. Each level of abstraction requires different capabilities and sophistication, while no layer of capability can exist correctly without the corresponding layers of functionality that lie above and below. Thus task decomposability is essential.

B. Control Loops for Robots and Humans

In executing this mission, a vehicle operator implicitly relies on a discrete decision process similar to the one graphically depicted in Figure 2. With this model, the operator periodically takes stock of the current situation, determines status of the current task, and proceeds to the next task when the current one is complete. In essence, applying a decision-loop model of Figure 2 to the mission description of Figure 1 transforms the description from a static overview of intended mission flow to an active specification that can be logically tested and mentally rehearsed prior to real-world execution.

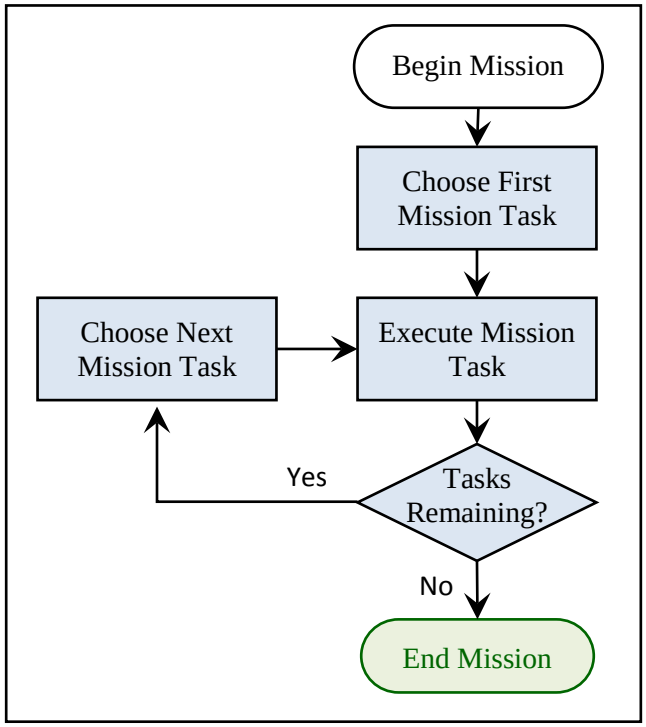

Figure 2. Supervisory task sequencer model for mission conduct as a series of discrete mission tasks and associated decisions.

In practice, this type of decision process is commonly referred to as a Sense-Decide-Act (SDA) loop when referring to overall control loops for computational autonomous agent activities, or sometimes a Sense-Interpret-Decide-Act loop when emphasizing machine evaluation of sensor inputs. Similarly, Figure 2 shows an Observe-Orient-Decide-Act (OODA) loop, commonly referred to when considering feedback control in military and other human operations (Hammond 2001) (Dannegger 2009) (Wikipedia 2016).

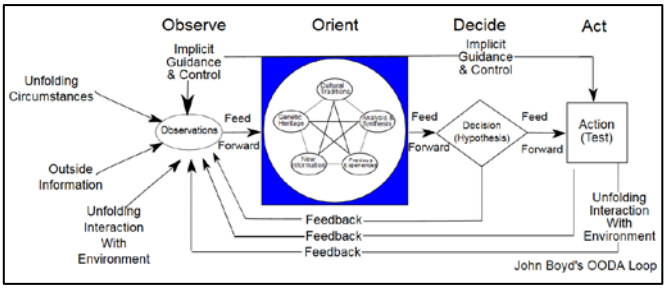

Figure 3. John Boyd's Observe Orient Decide Act loop describing human decision-making processes (Wikipedia 2016).

C. *Classical Decision Logic for Task Sequencing*

That similar SDA or OODA control-loop models can be applied to both human and non-human operators is noteworthy. It implies that missions thus specified might be executable not only by humans, but by human-controlled robots, human-robot teams, and suitably autonomous robots as well. One important aspect of the mission above must be accounted for, however. The sequential flow implicitly assumes success for each task. Where human operators are concerned, this is acceptable in most circumstances. When the ability to complete a task is in question, a human operator is able to request guidance from higher authority or use his best judgement to decide how to proceed. Under the requirements underpinning the framework proposed in this paper, this is not necessarily an option for robot agents. Rather, the course of action that the vehicle is to undertake in the event of task failure must be fully specified in the mission description. This can be achieved through the introduction of a simple branching structure.

As discussed previously, a specific autonomous agent may be trusted to execute a finite set of atomic behaviors that are used to define the mission. Further, the agent must be capable of detecting when a behavior is successfully completed and when the behavior cannot be successfully completed. It follows that a vehicle must be able to detect the success or failure of tasks within the mission definition so long as those tasks are comprised of trusted behaviors. This capability makes it possible to more rigorously define missions in a way that target autonomous vehicles can be trusted to execute without direct supervision.

With the introduction of potential branching based on task success or failure, overall mission success is no longer reliant on a fixed sequence of task executions. In fact, a particular mission can include the successful completion of some tasks, the failure of different tasks, and the complete omission of others. It is appropriate in this context to refer to the individual tasks as goals to be achieved rather than simply as tasks. A possible decision-tree elaboration of the mission from Figure 1 is provided in Figure 4. Interestingly, the SDA/OODA decision loop of Figure 2 is still suitable for controlling the execution of this revised mission.

Goal 1: Proceed to Area A and search the area. If the search is successful, execute Goal 2. If the search is unsuccessful, execute Goal 3.

Goal 2: Obtain an environment sample from Area A. If the sample is obtained, execute Goal 3. If the sample cannot be obtained, execute Goal 5.

Goal 3: Proceed to Area B and search the area. Upon either search success or failure, execute Goal 4.

Goal 4: Proceed to Area C and rendezvous with vehicle 2. Upon rendezvous success or failure, execute Goal 5.

Goal 5: Proceed to recovery position (mission complete). Upon successful arrival, mission complete. If unable to return to base, abort the mission.

Figure 4. Modified search and sample mission providing success-failure branching and human or autonomous agent execution (McGhee, Brutzman and Davis 2011).

This mission definition gives rise to a graphical understanding of mission flow, creating an alternate representation of the natural-language mission definition of Figure 4 above, which is next shown as the flow diagram of Figure 5.

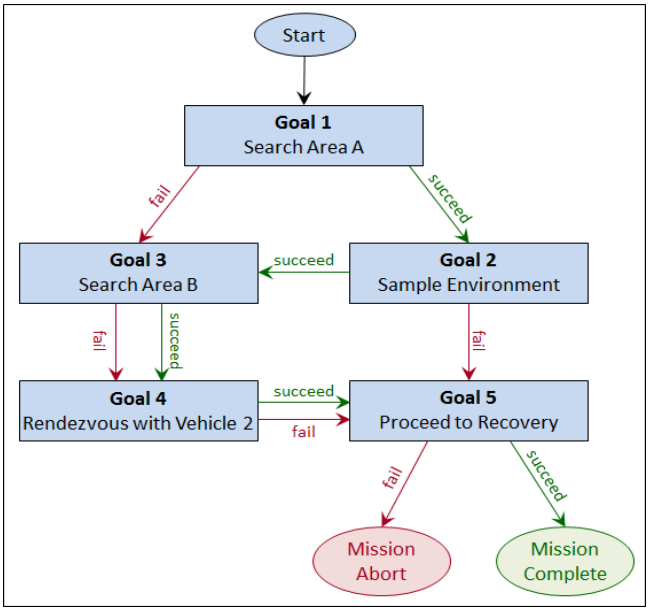

Figure 5. Mission-flow graph for a search and sample mission for human or autonomous agent execution (Brutzman, McGhee and Davis 2012).

This flow graph is one among many potential representational forms for this and many other missions, and a number of graphical, programmatic, and Extensible Markup Language (XML)-based definition forms have been proposed (Byrnes, et al. 1996) (Duarte, et al. 2005) (Davis, D. T. 2006). This flow-graph encoding is of particular interest because it provides an intuitive depiction of a potentially complex mission. In fact, an operator or supervisor can utilize a mission specification of this form to mentally "rehearse" the mission by intentionally traversing the graph from start to finish while exhaustively testing success and failure branches at every step. While not yet providing the required level of mathematical rigor, this ability to informally traverse all possible task sequences in this manner is an important step towards providing assurance to the responsible operator that the mission will progress according to human intent under all foreseeable circumstances.

D. Adding Constraints to Mission Decision Logic

As presented so far, this mission definition paradigm does not explicitly address the issue of ethical mission execution. Specifically, no mechanism has been suggested at this stage to define ethical constraints affecting the overall mission or individual tasks. It might be casually argued that ethical conduct is implied by "successful" completion of goal's requirements. Such an assumption is naïve, however, and does not provide nearly enough confidence for the operator to assume liability for the mission's conduct. For instance, it is apparent that an unmanned underwater vehicle (UUV) with an appropriate search behavior can achieve goals 1 and 3 of the example mission. Unfortunately, it may or may not be able to do so while avoiding detection, remaining clear of other vehicles in the area, or maintaining a specific navigational accuracy. If any of these (or other) conditional requirements must be met in order for the goal to be achieved in a safe and ethical manner, then an additional mechanism must be provided to incorporate those ethical constraints into the mission specification. In this context, ethical constraints do not describe characteristics of individual goals, but rather what must be considered and enforced during goal execution. From the standpoint of operator accountability, the constraints must be specified in a manner that preserves the ability to trace high-level mission flow, and also specified in a way that can ultimately be monitorable and enforceable by the autonomous vehicles themselves. A plain-language version of exemplar constraints is given in Figure 6.

Constraint 1:	The vehicle must maintain navigational accuracy within acceptable limits. Applies to entire mission.
Constraint 2:	All safety equipment must be fully functional. Applies to entire mission.
Constraint 3:	All mission systems must be operational. Applies to Goal 1, Goal 2, and Goal 3.
Constraint 4:	Acceptable distance from shipping lanes in the form of 1000 meter lateral standoff or minimum depth of 20 meters must be maintained. Applies to Goal 1, Goal 2, Goal 3, and Goal 4.
Constraint 5:	Must be able to detect surface contacts within 5000 meters. Applies to entire mission.
Constraint 6:	Detected surface contacts are to be avoided by a minimum of 1000 meters.

Figure 6. Constraints suitable for application to the example search and sample mission.

Ethical constraints vary and may be intuitively applied to either an entire mission or to relevant individual goals as appropriate. That is, there may be certain constraints that must be enforced from launch until recovery (e.g., all safety systems must remain operational), and others that only need to be enforced during the execution of specific goals (e.g., maintaining safety depth in the search area). As an example, the plain-language constraints of Figure 6 might be applied to the example UUV mission in a straightforward manner, as depicted graphically in Figure 7. Note that mission-level constraints 1, 2, and 5 could be applied to all goals individually without changing the constraint-application semantics. This construct still supports the prior rehearsal of missions for correctness, and also allows for the *in situ* consideration of whole-mission and goal-specific constraints (Brutzman, Davis, et al. 2013).

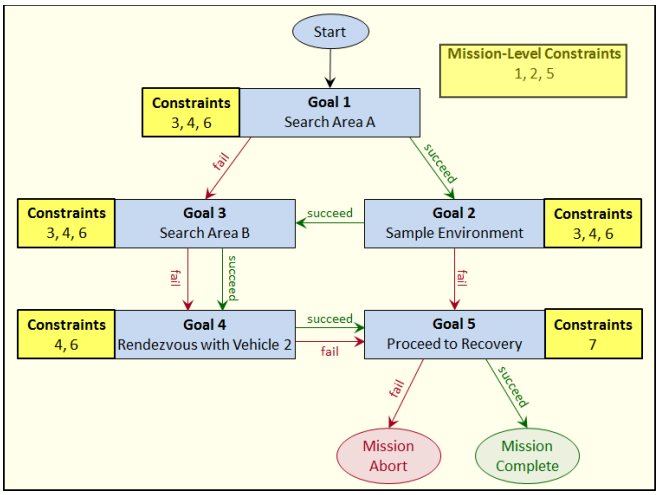

Figure 7. Mission-flow graph for a search and sample mission with ethical constraints applied as binary branching (D. P. Brutzman, D. T. Davis, et al. 2013).

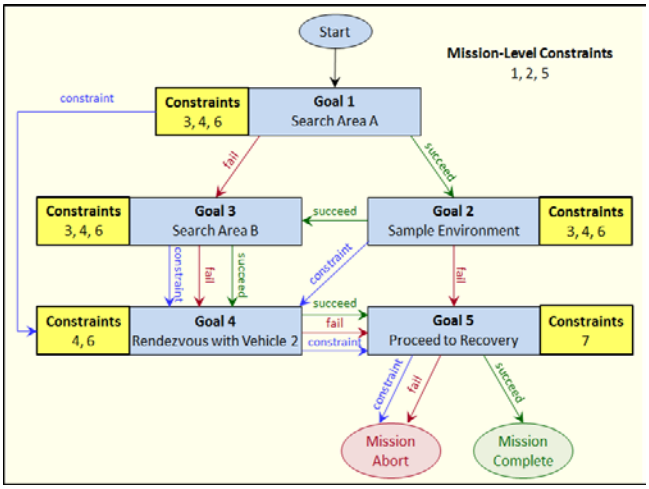

Figure 8. Mission-flow graph for a search and sample mission with ternary branching for imminent ethical-constraint violations.

Up to this point, the definition scheme only provides for binary branching of the mission-flow diagram: once initiated, a goal either succeeds or fails, and the mission then proceeds accordingly. Such a representation is fully representative of any decision tree, since tree graphs of arbitrary branching size can be traversed in a binary manner. However, a binary approach also presumes that an impending ethical constraint violation equates to goal failure. Such equivalence might be acceptable in many cases, and ethical violations causing goal failure certainly result in correct application of the constraints in the sense that goal execution no longer proceeds in the face of constraint violations. On the other hand, it might well be desirable to treat responses to impending constraint violations differently than simple failure (for example, in order to meet an additional independent objective). A more-responsive approach is possible through the addition of a third potential goal-execution outcome for constraint violations, along with a corresponding branching option in the mission flow structure. That is, execution of an individual goal becomes terminated upon goal success, goal failure, or impending violation of a constraint applied to that goal. Flow of control then proceeds as directed to whichever subsequent goal is next designated as appropriate.

This constraint-based tree approach shown in Figure 8 is quite useful and an excellent match for supervisory planning needed when humans perform robot mission planning. The general expressive power of binary-flow logic is preserved, and system responses must be explicitly considered for each success, failure, or constraint violation that might occur when performing each mission goal. The ternary-flow structure is also similar to exception handling in modern programming languages, which can facilitate implementation and testing. The corresponding modification of the example mission to illustrate this ternary branching model is graphically depicted in Figure 8.

E. Implications of Ethical Constraints on Mission Tasking

Designing robot missions in the form of a flow diagram consisting of a set of discrete goals, with ethical constraints applied to individual goals as described here, provides an intuitive mechanism that can enhance responsible operators' understanding of the missions they expect to supervise. The nature of the mission specification is declarative. At this level of abstraction, individual goals execute sequentially according to the mission tree, irrespective of elapsed time, and each goal predictably terminates in one of three possible states (goal success, goal failure, or constraint violation).

Supervisory trust that a directed vehicle can execute specific goals, recognize goal failure, and identify pending constraint violations provides important boundaries on autonomous behavior. Essentially this approach eliminates any need to make assumptions or guesses concerning intended vehicle conduct during goal execution. Rather, the onus of well-specified tasking is specifically placed on human operators to create well-defined and thorough missions. Further, if the size of the mission-flow diagram is reasonably managed, then exhaustive testing of all possible mission execution sequences is achievable and tractable. These aspects of mission design are fundamentally important, and are essentially quite similar to the essence of coordinated operational tasking among ships and aircraft led by responsible and cooperating humans.

The next section strengthens the foundations for these concepts. Examining the underlying nature of the mathematical formalizations used here can provide further operator assurance that a particular mission is appropriately defined and can proceed as expected, in an appropriate matter, under all circumstances.

V. Mission Execution Automata (MEA): Executable Mission Specifications

A. The Rational Behavior Model (RBM) Robot Control Architecture

Autonomous vehicle mission expression in the form of a flow diagram is compatible with the higher levels of abstraction for a number of proposed hierarchical robot control architectures (Byrnes, et al. 1996) (Ricard and Kolitz 2002) (Albus 1998). One particularly relevant example is the Rational Behavior Model (RBM) tri-level software architecture (R. Byrnes 1993). A variety of other 3-level robot architectures have proposed and implemented over the past two decades, typically with similar timing principles and varying jargon. RBM is modeled on the command hierarchy of a manned submarine, and organizes robot control requirements into execution, tactical, and strategic levels as depicted in Figure 9.

- *Execution-level control* includes those hard-real-time uninterruptable tasks associated with control and management of hardware systems that direct interact with the vehicle's physical environment. These feedback-driven controllers correspond to the activities of a manned vessel's junior crew-members and include manipulation of control surfaces and sensors.
- *Tactical-level tasks* utilize execution-level functionality to realize more complex behaviors. Tactical-level task behaviors correspond to those activities managed by a manned vessel's watch officers and can be as simple as maintaining a desired course and speed or transiting to an ordered geographic location or arbitrarily complex, such as conducting an area search or utilizing onboard sensors for map development.
- *Strategic-level goals* are at the highest level of control and correspond to the activities directed by a manned vessel's commanding officer. These goals control overall mission conduct by initiating tactical-level behaviors as prescribed by the mission definition.

The previously described mission-flow diagram and performance of each mission's individual goals align well with the RBM Strategic and Tactical Levels respectively. Within this model, the Strategic Level operates in a discrete manner completely in the mathematical realm of formal logic. Once a Tactical-Level behavior is activated, Strategic-Level execution pauses until a response is received from the Tactical Level indicating that the behavior (goal) was successfully completed, failed to complete, or encountered a constraint violation.

If properly encoded, the Strategic-Level mission-flow diagram can actually form an executable mission specification. Declarative forms that are both human-readable and machine executable versions of the exemplar mission have been developed with the Prolog language and XML (McGhee, Brutzman and Davis 2011) (Brutzman, McGhee and Davis 2012) (Brutzman, Davis, et al. 2013) (as well as in an analytical combat simulation as discussed below). This human-and-machine compatible form is in line with the ethical framework requirements listed earlier and provides for human-based testing of mission code prior to robot (or joint robot-human) execution. Further, from the perspective of the Strategic Level, it does not matter whether the Tactical Level behavior is executed by an actual robot, a computational model, or a human being. Thus, Tactical-Level responses provided by a human tester are not only analogous to those provided by an actual robot—they are identical to actual robot responses for the purposes of testing high-level mission response.

B. Mission Execution Automaton (MEA) Definition

Mathematical rigor of Strategic-Level execution steps is obtained by observing that the run-time traversal of the mission-flow diagram is similar to the operation of a mathematical formalism called a Turing Machine (TM). A Turing Machine is a Finite State Machine (FSM) with an associated one-dimensional infinite storage medium called a tape (Minsky 1967). TMs have a number of fundamental properties that are

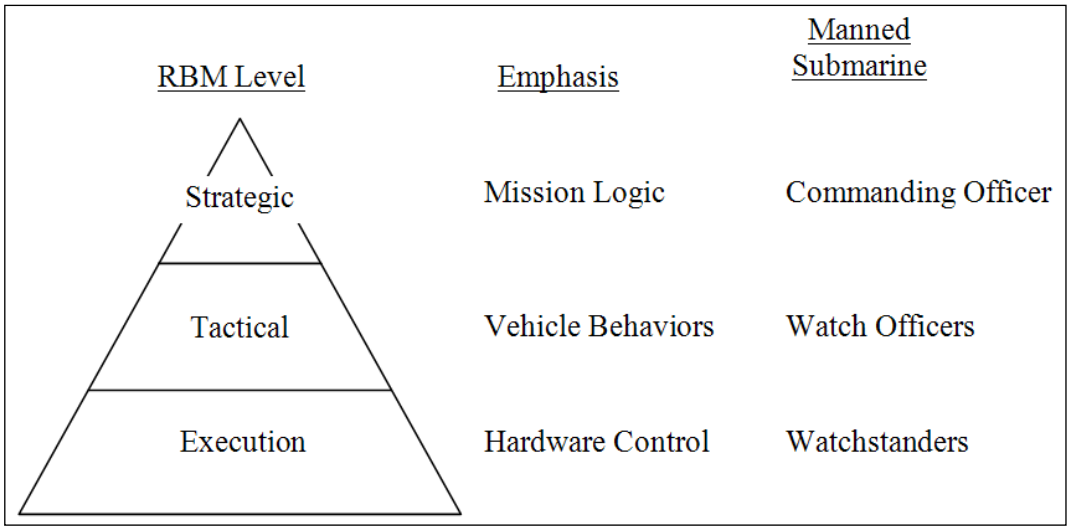

Figure 9. The Rational Behavior Model (RBM) software architecture based on the hierarchical control paradigm of naval vessels (R. Byrnes 1993).

```
Ethical Mission Execution Trace #1:
?- execute_mission.
Commence: Search Area A.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? constraint.
Commence: Rendezvous with vehicle 2 in Area C.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? constraint.
Commence: Return to Base.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? constraint.
Mission Abort!

Ethical Mission Execution Trace #2:
?- execute_mission.
Commence: Search Area A.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? succeed.
Commence: Take environmental sample from Area A.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? succeed.
Commence: Search Area B.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? succeed.
Commence: Rendezvous with vehicle 2 in Area C.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? succeed.
Commence: Return to Base.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? succeed.
Mission Complete!
```

Figure 10. Typical execution traces of Strategic Level testing of the mission depicted in Figure 7 using MEA Prolog source code (**bold** indicates operator input).

particularly important in the field of computer science, most notably that the computational power of a TM is equivalent to that of a digital computer (Minsky 1967) (Petzold 2008). Despite this expressive equivalence, TM "programming" to effect useful computation is generally considered impractical, and TMs have been relegated largely to academic study (McGhee, Brutzman and Davis 2011). They do, however, provide a strong theoretical foundation upon which to build a mathematically sound Strategic-Level mission-flow diagram execution mechanism suitable for both robots and human operators.

A useful modification to the basic TM definition is to separate FSM operations from the underlying computational mechanism. Referred to as a Universal Turing Machine (UTM), this variation provides a TM capable of operating with an arbitrary FSM (Minsky 1967) (Petzold 2008). With this modification, the TM execution mechanics can be viewed as the "computer", and individual FSMs as "programs." More specifically, the mission-flow diagrams described here can be viewed as robot programs (they are a form of FSM) that are executable using essential UTM semantics.

Utilization of the UTM concept requires one additional abstraction: since Tactical Level responses are not read from a (so-called) TM tape input, the UTM definition must be extended to allow inputs from one or more (human or robot) agents. More formally, the theoretical UTM tape definition must be generalized to any type of external agent with a finite set of input and output symbols and arbitrary additional capabilities. It can be demonstrated that abstracting the tape portion of the UTM in this way is, in fact, a generalization that subsumes the more constrained UTM (McGhee, Brutzman and Davis 2011). Thus, from a computational standpoint, the expressive power of this result can be considered "Turing complete" and capable of executing any robot mission defined in the form of a flow diagram. This UTM generalization is referred to here as a Mission Execution Automaton (MEA) which consists of a Mission Execution Engine (MEE) and an arbitrary set of mission orders in FSM form (McGhee, Brutzman and Davis 2012).

Formally, we define the MEA as follows:

$M = (Q, \Gamma_i, \Gamma_r, b, \delta, \gamma, q_0, F)$,

where:

Q = a finite, non-empty set of *states*

Γ_i = a non-empty set of input symbols corresponding to *behavior-initiation function calls* to the tactical level (note, once parameterization of function calls is taken into consideration, this set is of potentially infinite size but is practically constrained to a finite set by γ)

Γ_r = a finite, non-empty set of *response symbols* corresponding to *return values* from behavior function calls

$b \in \Gamma_i$ is a *blank* and equates to no function call (note, since no function call is made, no response will be received, so execution will halt in the current state)

δ: $(Q \setminus F) \times \Gamma_r \rightarrow Q$ is the *transition function* mapping a current state and response to a new state

γ: $Q \rightarrow \Gamma_i$ is the *behavior call function* that maps a state to a behavior-initiation function call

$q_0 \in Q$ is the initial state

$F \subseteq Q$ is the set of final states which equate (if the mission is constructed correctly) to mission termination

Constraints can be added explicitly to the definition by adding a set of constraints C, a constraint-mapping function τ: $Q \rightarrow c \subseteq C$, and modifying Γ_i and γ to account for τ.

C. Strategic-Level Mission Rehearsal and Testing

MEA implementations were initially developed in Lisp and Prolog (Brutzman, McGhee and Davis 2012) (McGhee, Brutzman and Davis 2011) (McGhee, Brutzman and Davis 2012). This decision was made because of extensive experience of the authors in using Prolog at the RBM Strategic Level (R. Byrnes 1993) and because of the strong simulation capability of these languages. This latter characteristic is made evident by an MEE Prolog software implementation. As can be seen, this code requires only 14 lines of Prolog code, five of which are dedicated to human external agent communications functions (analogous to TM inputs and outputs) (Brutzman, McGhee and Davis 2012). Further, mission orders corresponding to the mission depicted in Figure 7 are encoded in only 28 lines of Prolog. In addition, the declarative nature of Prolog enables the definition of mission orders that strongly resemble English (with a small amount of specialized punctuation and semantics), making Prolog mission orders intuitively understandable by non-programmers. This working Prolog simulation allows testing of the Strategic Level mission flow by a human operator, with typical results depicted in Figure 10. In the simulation, the Strategic Level orders commencement of individual goals and the human operator reports success, failure, or constraint-based termination of each goal. The depicted traces in Figure 10 correspond to instances where first each goal terminates due to potential constraint violation (Trace #1) and then where each goal completes successfully (Trace #2).

Strengths of the Prolog implementation notwithstanding, it is important to note that there is nothing inherently unique about Prolog execution, and the MEE and mission orders can be accurately created with any Turing-complete computer language. Successful implementations have been developed by the authors in the Java programming language (Davis, D. T. 2006) (Brutzman et al. 2016). A similarly capable, independent implementation uses the Hierarchical Task Network (HTN) behavior model and Python programming language in the Combined Arms Analysis Tool for the 21st Century (COMBATXXI), a simulation tool developed and used by the U.S. Army and U.S. Marine Corps within various analytic studies (U.S. Army Training and Doctrine Command Analysis Center 2015) (Posadas 2001). Results employing a COMBATXXI implementation corresponding to those shown in Figure 10 are presented in Figure 11.

For this implementation, a Python class represented the structure of the mission graph in Figure 7. The content of the graph was encoded in an instance of that Python class.

Based on this experience, as well as other results using the XML-based AVCL mission specification (Davis, D. T. 2006) (Brutzman, Davis, et al. 2013) (Brutzman et al. 2013), it is reasonable to conclude that flow graphs represent a higher level of abstraction for mission specification than any unconstrained text-based programming language. Moreover, advances in graphical coding (Langley and Spenser 2005) may eventually allow non-programmers to completely specify robot missions by constructing a flow graph such as Figure 7 directly on a computer screen. Such advances can further enhance the comprehension, supervision and accountability of mission experts for producing legally valid mission definitions (and avoiding mission errors).

D. Progressive Refinement of Complex Mission Tasks

Referring to our exemplar mission in Figure 7, it is implicit that Goal 1 is achievable only if the person or software at the Tactical Level has considerable knowledge about Area A and how to search it. To make this concrete, suppose that Area A contains hazards that are potentially harmful to (or impassible by) the search vehicle. Also suppose that the tactical officer's supporting software has no reliable or current map for the area of interest. A classic algorithm for exploring for such circumstances is depth first search (Skiena 1997). Such a search continues until the object of the search is located (success) or the area has been completely searched without finding the search objective (failure). The search proceeds by moving the vehicle into an accessible terrain cell and testing to see if the goal is there. If not, then the vehicle moves forward to a previously unexplored cell, and again checks for presence of the goal. If no such cell exists, then the vehicle retreats to the previous cell (backtracks) and marks the cell just visited as a "virtual obstacle."

Figure 12 depicts a flow graph for the above-described process, and further represents a refinement solely for Goal 1 of mission Figure 7. For this reason, subgoals within this refinement of Goal 1 have been labeled 1.1, 1.2, etc. It is possible to interpret Figure 11 in several ways. First of all, if trusted vehicle behaviors corresponding to each of the goals in this figure exist, they might also be commanded by a human operator acting on behalf of the Tactical Level. An example of interactive testing of this possibility is illustrated in Figure 13. Alternatively, this depiction may be understood as a Tactical Level implementation capable of autonomous execution of Goal 1 from Figure 7. Note that constraint checking on Goal 1.1 is not a precondition *per se*, rather all constraints must be met continuously by all sub-goals once Goal 1 has commenced.

Frequently applying the double-check question "how might a human accomplish this task?" is an important design principle for autonomous-system mission production. This task decomposition raises important additional implementation questions, namely, are the semantics of a depth-first search accurately represented by Figure 13 and then correctly encoded in the supporting Tactical Level code? Unfortunately, these questions cannot be conclusively answered by exhaustive testing as was possible for the strategic level mission orders

Ethical Mission Execution Trace #1:

```
Commence: Search Area A.
Did goal Succeed, Fail, or end with a Constraint? constraint.
Commence: Rendezvous with Vehicle 2.
Did goal Succeed, Fail, or end with a Constraint? constraint.
Commence: Proceed to Recovery.
Did goal Succeed, Fail, or end with a Constraint? constraint.
Mission Abort!
```

Ethical Mission Execution Trace #2:

```
Commence: Search Area A.
Did goal Succeed, Fail, or end with a Constraint? succeed.
Commence: Sample Environment.
Did goal Succeed, Fail, or end with a Constraint? succeed.
Commence: Search Area B.
Did goal Succeed, Fail, or end with a Constraint? succeed.
Commence: Rendezvous with Vehicle 2.
Did goal Succeed, Fail, or end with a Constraint? succeed.
Commence: Proceed to Recovery.
Did goal Succeed, Fail, or end with a Constraint? succeed.
Mission Complete!
```

Figure 11. Human interaction with a Strategic Level mission results obtained from the COMBAT XXI simulation.

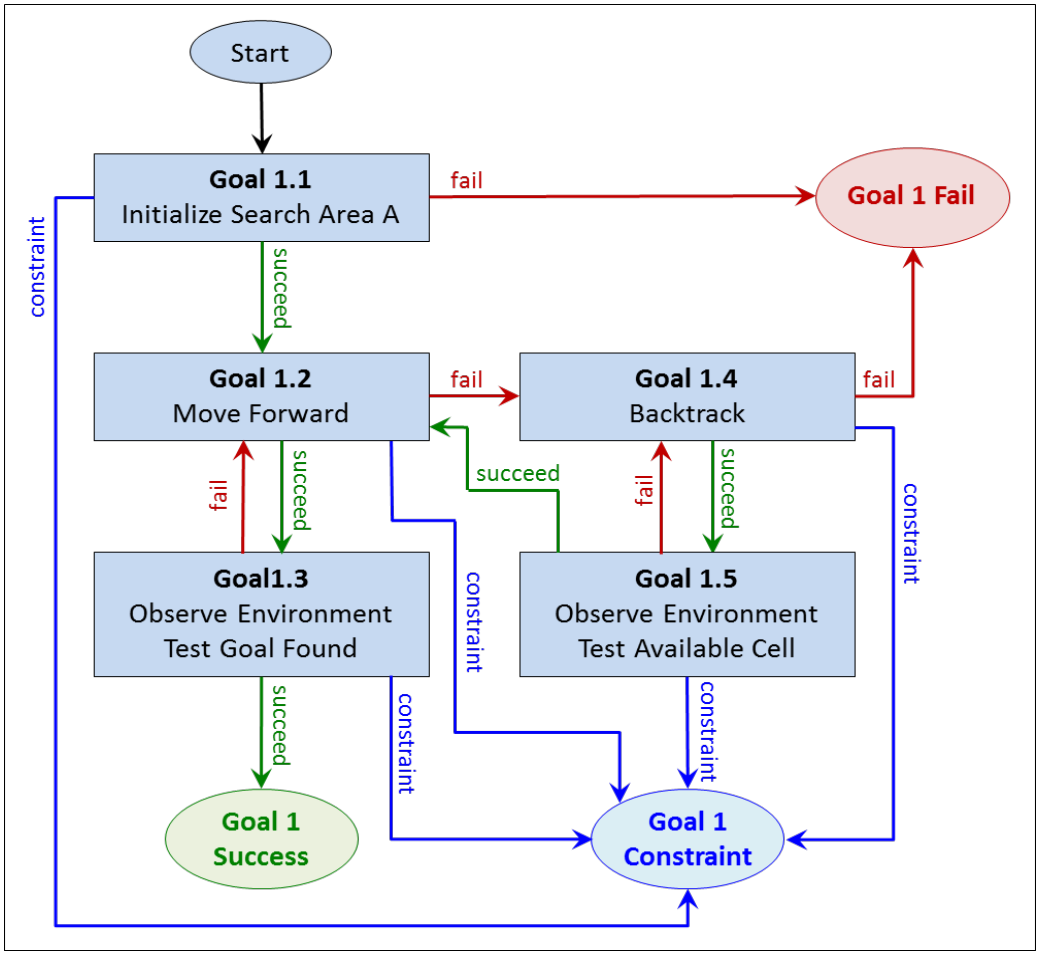

Figure 12. Progressive refinement, illustrated. Flow Graph for a grid-based depth-first search of Area A corresponding to subgoals within Goal 1 of Figure 7, adapted from (McGhee, Brutzman and Davis 2012).

This limitation on testing occurs because the individual Goal 1 in Figure 12 contains several loops and, in general, there is therefore no guarantee that it will terminate for any terrain, much less be tested exhaustively. Moreover, because of the need to remember the location of virtual obstacles, a potentially infinite memory is required to execute the algorithms. Thus, Figure 12 actually defines a Turing Machine and it is known that, in general, the correctness of a Turing Machine represented in the graph cannot be proved. Fortunately, it is known that for a finite search area, depth first search will eventually terminate with either success or failure in searching for a specified goal (Skiena 1997). Nevertheless, exhaustive testing for all possible terrain samples is not possible. Instead, the most that can be asked for is to show that all phase transitions in the given flow graph are correctly coded. Careful examination of the corresponding algorithm logic in Figure 12 shows that the included examples do in fact test all possible control branches, and that they have been correctly implemented.

It turns out that the inability to exhaustively prove correctness of a flow graph at the Tactical Level is not quite as serious as it might at first seem. This is because algorithm failure at this level amounts to just one more reason that might cause a phase of the Strategic Level to fail. Since every Strategic Level failure is accounted for in the overall mission flow graph, the mission can still continue in a planned way when such cases occur. To make absolutely sure that such dependability occurs, real-world clock time available at the Tactical Level (R. Byrnes 1993) must be used to create a time-out condition resulting in goal failure for any process that continues beyond a specified maximum execution time duration. In addition, it is expected that the Tactical Level continues to observe all constraints that are applicable at the Strategic Level throughout the performance of a particular goal. Thus in the case of Figure 13, constraints one through six from Figure 5 apply throughout the entire period of tactical-level testing as well.

When enough experience has been obtained by manual execution of the mission specification with Tactical Level decisions made by a human operator, the mission can be encoded to allow autonomous mission execution. This is accomplished in (McGhee, Brutzman and Davis 2012) through the addition of robot communication functions to the MEE definition. As an example, typical results of an updated version of this implementation are presented in Figure 14. For a more complete understanding of an actual mission log's importance, it should be recognized that in the real world, for a fully autonomous mission, such information may only be available if the vehicle succeeds in returning to the original intended recovery position. Complete loss of event logs and telemetry data prior to catastrophic loss is a common occupational hazard for unmanned systems at sea. However, in simulation form, such test results can be available during pre-mission testing to an observer who can, if necessary, use such diagnostics to revise mission orders or Tactical Level robot software before actual mission execution.

Evidently, commands from the Strategic Level must directly invoke trusted Tactical Level behaviors or utilize goal refinement as outlined above to enable Tactical Level execution of the mission flow graph. Thus, for instance, the Tactical Level behavior invoked to achieve Goal 3 ("Search Area B") must be entirely self-contained (i.e., atomic) unless it is in the form of flow-graph-connected atomic behaviors. This requires that any prior knowledge of the characteristics of this area be taken into account.

For the example being considered, if it is known that Area B contains no hazards or obstacles to vehicle motion, a rectangular "lawn mower" search pattern may be appropriate. Such decisions may be difficult to make without a great deal of knowledge concerning a given area of operations for a mobile robot. One effective way of achieving effective mission planning at both the Tactical and Strategic levels is to incorporate all available knowledge concerning the vehicle and its area of operation into a detailed physical-model-based computer simulation. One such simulation system, the AUV Workbench (Brutzman et al. 2016), can be used to present detailed examples and associated graphical display such as those presented in (Brutzman, Davis, et al. 2013).

A final observation concerning Tactical Level mission software is that available behaviors can be combined by "task abstraction" to produce ever higher-level trusted behaviors until the commands from the Strategic Level can be directly executed as function calls. That is, once a flow graph accomplishing a specific purpose (e.g., depth-first search) has been suitably vetted, it effectively becomes a trusted Tactical Level behavior itself. This has proven to be an effective means of incrementally increasing autonomous capabilities. This means reinterpreting Tactical Level flow graphs as code specifications rather than actual code to be executed. This alternative has been implemented and tested using Allegro Common Lisp in (McGhee, Brutzman and Davis 2012).

```
Depth-First Search Execution Trace #1:
?- execute_goal.
Commence: Initialize Search Area A.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? succeed.
Commence: Move Forward.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  succeed.
Commence: Observe Environment, Test Goal Found.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  fail.
Commence: Move Forward.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  fail.
Commence: Backtrack.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  succeed.
Commence: Observe Environment, Test Available Cell.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  fail.
Commence: Backtrack.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  fail.
Depth First Search:  Reporting Goal Failure!.

Depth-First Search Execution Trace #2:
?- execute_goal.
Commence: Initialize Search Area A.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? succeed.
Commence: Move Forward.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  succeed.
Commence: Observe Environment, Test Goal Found.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  fail.
Commence: Move Forward.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  fail.
Commence: Backtrack.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  succeed.
Commence: Observe Environment, Test Available Cell.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  succeed.
Commence: Move Forward.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  succeed.
Commence: Observe Environment, Test Goal Found.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)?  succeed.
Depth First Search:  Reporting Goal Success!.

Depth-First Search Execution Trace #3:
?- execute_goal.
Commence: Initialize Search Area A.
Did goal Succeed (s), Fail (f), or end with a Constraint (c)? constraint.
Depth First Search:  Reporting Constraint Encountered!
```

Figure 13. Execution trace examples of human-supervised depth-first search of Area A using Prolog MEA implementation shown in Figure 14. (**bold** indicates operator input) (adapted from (D. P. Brutzman, D. T. Davis, et al. 2013)).

```
Autonomous Mission Execution Trace #1:
?- auto_execute_mission.
Commence: Search Area A.
Goal execution constraint terminated!
Commence: Rendezvous with vehicle 2 in Area C.
Goal execution succeeded!
Commence: Return to Base.
Goal execution constraint terminated!
Mission Abort!


Autonomous Mission Execution Trace #2:
?- auto_execute_mission.
Commence: Search Area A.
Goal execution succeeded!
Commence: Take environmental sample from Area A.
Goal execution failed!
Commence: Return to Base.
Goal execution succeeded!
Mission Complete!

Autonomous Mission Execution Trace #3:
?- auto_execute_mission.
Commence: Search Area A.
Goal execution failed!
Commence: Search Area B.
Goal execution succeeded!
Commence: Rendezvous with vehicle 2 in Area C.
Goal execution failed!
Commence: Return to Base.
Goal execution constraint terminated!
Mission Abort!
```

Figure 14. Examples of mission log for simulation of fully autonomous execution of the mission defined in Figure 7 (**bold** indicates operator input), adapted from (McGhee, Brutzman and Davis 2012).

E. Summary of Insights, Mission Execution Automata (MEA)

As a generalization of the Turing Machine, the MEA provides a mathematically sound approach to the definition and exhaustive testing of unmanned vehicle missions. The MEA includes a mission-specific FSM and unlimited memory. Consistent with the MEA generalization, the TM tape can be replaced by a physical robot or a human being to which output can be sent (commands) and inputs can be received (e.g. success, failure, or constraint-violation responses). Other external agents can be optionally added as well.

In order to guarantee eventual termination of a mission, the structure of a Strategic Level mission must be constrained somewhat beyond the basic MEA definition. Specifically, the mission FSM cannot include loops, unreachable states, or sink states (i.e., non-terminal states from which further transitions are not possible). Further, the Strategic Level mission must be defined with few enough states and transitions to allow for tractable exhaustive testing by a human operator. However, when a Strategic Level goal is iteratively refined to develop a Tactical Level behavior (as with the depth-first search example) these restrictions do not apply since the Tactical Level can implement a time-out failure to ensure termination of individual behaviors.

Finally, a universal MEA can be achieved by implementing sequencing and communication functions as a separate MEE and then developing the mission flow graph as a set of mission orders in a form understandable by both the MEE and humans who are mission specialists, but who may not be programmers. There are many choices for expressing such mission orders including flow charts, text-based programming languages, and graphical user interface techniques.

VI. Validation of RBM Software Architecture through Real-World and Virtual Experimentation

Up to this point, all results presented have related to the Strategic Level and the Tactical Level of RBM software for a single example of a "search and sample" mission for a notional autonomous underwater vehicle (AUV). Furthermore, all results presented thus far have been obtained from high-level mission simulations written, except for Figure 11, in the Prolog logic programming language. However, beginning in 1993, in parallel with formalization and publication of details of RBM (Byrnes, et al. 1996), the authors and their collaborators demonstrated the value and practicality of this approach for undersea robots through a series of open-ocean experiments involving two small unmanned submarines. These experiments and results obtained are summarized in the following subsections.

A. Phoenix Autonomous Underwater Vehicle (AUV)

The Phoenix AUV was an unmanned submarine designed and built at the Naval Postgraduate School (NPS) beginning in 1990. About six feet in length and weighing approximately 500 pounds, Phoenix included four cross-body thrusters, enabling active control of five degrees of motion (x, y, z, pitch and yaw) (Ortiz and Šimkus 2012). Interestingly, roll motion was also controllable during forward motion when making a turn. In any case, such flexible control through multiple degrees of freedom enables maneuvering in tight spaces comparable to the capabilities of a helicopter. This maneuverability allowed early Phoenix testing in a swimming pool, and later in a special test tank which provided greater depth for vehicle motion and control (Marco, Healey and McGhee 1996).

Phoenix' real-time control software was developed in a bottom-up fashion starting with control of maneuvering planes and thrusters on a timed interrupt basis in response to sensory input from an on-board sonar system along with depth, water-speed, and heading sensors. Commanded motions were monitored by a human operator observing the submarine in the test tank and connected by a floating network cable. Simultaneously, high-level Strategic Level mission-control software was developed in Prolog through the use of testing via computer simulation. Large numbers of high-fidelity physics-based simulations were needed to correctly develop and test what were then considered AI approaches to replace human supervision. While this simulation involved distinct mission phases in the form of a command "script" similar to Figure 1, no binary flow graph with phase-failure contingencies was abstracted from these phases as described in the preceding section of this paper. This meant that exhaustive testing of a mission was not possible, and there was therefore no proof of correctness available. Moreover, other than necessary supervision and safety monitoring, no concept of ethically constrained behavior was attempted in any of this work.

To achieve proper sequencing of phases, the Phoenix system made use of the backward chaining theorem-proving capability of Prolog (Rowe 1988). Specifically, a mission was initiated and carried out by issuing a query to the top level of Prolog asking (adapted from Prolog syntax): "Using the given mission rules and initial facts, along with a set of mission goals, are there any variable bindings that make execution of the given mission script possible?" A negative result terminated the mission in an explicit abort command. This mode of operation obviated the need for an MEE, but did not allow branching on phase failure as in the simulation studies described in the preceding sections of this paper.

During tank testing, the Phoenix was initially operated using an onboard "Gespac" real-time control computer for tactical and execution level functions. This simple computer was not able to host the backward-chaining Prolog program running the goal-driven Strategic level. Therefore, before at-sea testing, an onboard "Sparc" UNIX computer hosting Prolog was added to implement the Strategic Level and connected to the Gespac computer by an Ethernet cable (Ortiz and Šimkus 2012). Further testing was then carried out to validate the correct functioning of these two computers together to execute simple missions in the test tank (Marco, Healey and McGhee 1996). Finally, before commencing open-ocean testing in Monterey Bay, a full strap-down inertial navigation system employing GPS and water-speed sensing for drift correction was added to the onboard electronic suite of the Phoenix (Ortiz and Šimkus 2012). Following successful at-sea testing, results obtained were used to design a larger vehicle, the Aries AUV, described in the next section.

B. Aries AUV

Aries was a somewhat larger vehicle than Phoenix, with a similarly rectangular hull, and was specifically designed for open-ocean surveys (Davis, D. T. 2006). It therefore used more efficient forward thrusters and lacked cross-body thrusters. This meant that extensive test tank debugging of strategic level code was not possible. Instead, a detailed and accurate physically based model of the vehicle and its environment, with three-dimensional (3D) real-time graphical display, the Autonomous Unmanned Vehicle (AUV) Workbench, was developed and used for real-time testing of robot mission software (Brutzman, Davis, et al. 2013) (Brutzman, et al. 1998) (Brtuzman 1994) (Brutzman et al. 2016).

Because of the cumbersome nature of the two-computer onboard control system used in Phoenix and the software complexity of issuing C function calls from Prolog, it was decided to use a single computer on Aries, and to write a custom onboard task sequencer as represented by Figure 2. Furthermore, realizing the desirability of branching on task failure but lacking as yet the concept of an MEA, a graphical user interface (GUI) was created to allow users to define phases and corresponding phase successors for strategic level mission definition and execution (Brutzman, Davis, et al. 2013). At this time, however, the possibility of ternary branching to account for ethical or safety-related constraint failure had not occurred to the authors, and it was therefore not included in Aries programming tools or experiments.

Aries AUV missions were defined with AVCL, a schema-constrained XML data model supporting autonomous vehicle mission definition, execution, and management (Davis, D. T. 2006). While the mathematical concept of an MEA had not

been developed at the time of AVCL's development, AVCL does provide a fixed set of goal types including area search, environmental sampling, and rendezvous and is thus suitable for the definition of mission flow diagrams such as the one depicted in Figure 4. Further, AVCL was intentionally designed to support implementation of the RBM Strategic and Tactical Levels and was utilized to define RBM-controlled Aries missions for simulation in the AUV Workbench virtual environment and for open-ocean real-world tests.

As an example, consider the XML snippet of Figure 15 which provides a hypothetical description of Goal 1 from Figure 7 for execution by an unmanned underwater vehicle. This specification defines the type of search to be conducted (area search for multiple targets with an expected probability of detection of 0.8), the area to be searched (a 500 meter by 3000 meter rectangular area with a northwest corner at 36.7 north latitude and 121.9 west longitude), and stipulates that the search be conducted at a depth of between 25 and 50 meters. Evidently, the search goal definition describes what is required for successful completion of the goal. It does not, however, dictate precisely how the goal is to be completed since such navigation and maneuvering decisions are left to the Tactical-Level implementation.

Simulation of a mission consisting an AVCL specification for a search goal similar to the one in Figure 15 and avoid areas specified as constraints in the AUV Workbench is shown in Figure 16. During the mission, the Tactical Level plans a path and maneuvers to the search while remaining clear of the avoid areas and then develops and executes a suitable pattern for the required area search. More complicated missions demonstrating the binary branching model were conducted in AUV Workbench simulations and also in open-ocean experiments in Monterey Bay (Davis, D. T. 2006) (Brutzman et al. 2016). A comprehensive comparison and consolidation of goal types can be found in (Davis, D. T. 2006).

```
<Goal  description="search operating area A" id="Goal1" >
    <Search  datumType="area" requiredPD="0.8" singleTarget="false" />
    <OperatingArea>
        <Rectangle>
            <NorthwestCorner>
                <LatLonPosition  latitude="36.69" y="-121.90" />
            </NorthwestCorner>
            <Width  value="500.0" />
            <Height  value="3000.0" />
        </Rectangle>
        <DepthBlock  minimum="25" maximum="75" />
    </OperatingArea>
</Goal>
```

Figure 15. An Autonomous Vehicle Command Language (AVCL) specification of Goal 1 from Figure 7 for execution on the NPS Aries unmanned underwater vehicle (Davis, D. T. 2006).

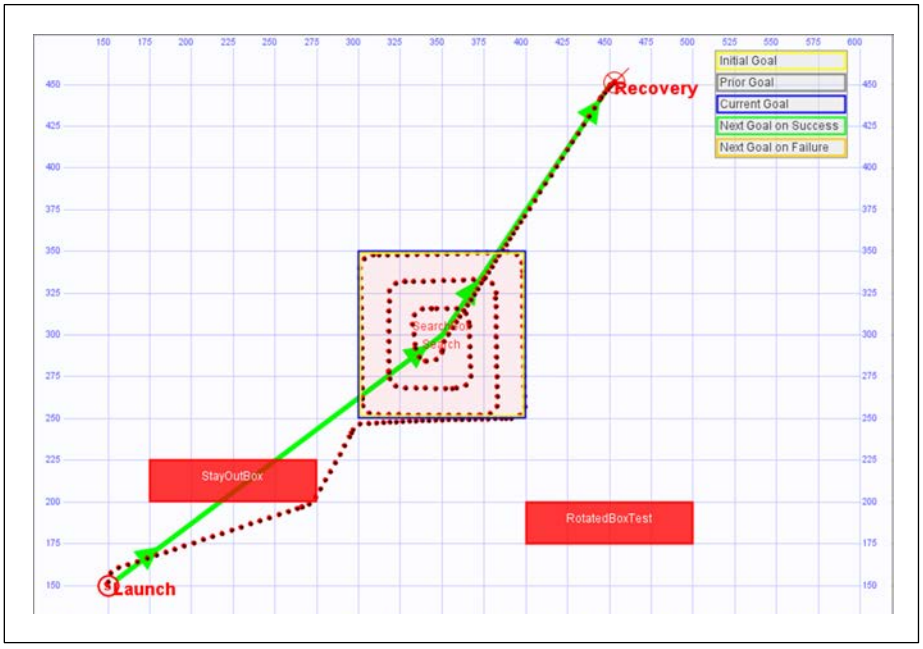

Figure 16. AUV mission SimpleBoxTest.xml written in AVCL that demonstrates simulated conduct of a goal-oriented mission that was performed amidst constraints. (Brutzman, et al. 1998)

C. Dangers Associated with Using Rules and Fact Assertion to Implement Strategic-Level Logic

As described above, Phoenix missions were executed as a result of an inferencing and reasoning process, using a set of rules and facts for mission definition. While all in-water missions succeeded, and Phoenix was never lost at sea, this approach provided no means of proving the correctness of Strategic Level software comparable to the exhaustive testing made possible by MEA formalism. The authors believe that this is a serious limitation that applies to all approaches for top-level Strategic Level mission definition that require specific actions to be derived from general principles rather than using a completely concrete finite-state machine (FSM) approach.

Specifically, a set of rules and facts amounts to a formal mathematical system in which the rules and facts serve as axioms. Theoretically it is known that, in general, no such set of axioms can be proved complete. Here, completeness means that all true theorems can be proved by formal application of predicate calculus. Such properties are very hard to prove. In fact, to the astonishment of the entire mathematical world, Gödel proved in 1931 that such a simple system as integer arithmetic cannot have any axiomatic basis. Perhaps equally shocking, even plane geometry had no sound axiomatic basis until around fifty years ago. This meant that, from a strictly formal perspective, all of Euclid's original "proofs" were merely plausibility arguments. Fortunately, all of the theorems believed to be true are in fact provable using the complete and consistent set of modern algorithms (Simpson 2000).

The significance of the above observations relative to top level mission specification derives from the fact that, for rule-based systems, mission execution can sometimes be regarded as a side effect of proving the theorem that "there exists a way to satisfy all specified mission goals while observing all given constraints." If it eventually turns out that the mission axiom set contains a contradiction, then system execution behavior becomes unpredictable and not testable, as well as potentially hazardous and even self-defeating.

The potential unpredictability of less-formal reasoning approaches and the inability to prove the correctness and completeness of axiomatically defined missions effectively precludes formal responsibility or liability for robot missions using these approaches. In fact, artificial intelligence (AI) approaches to top-level mission specification and control almost invariably make use of some form of reasoning and/or statistical pattern recognition. Applying such broad abstractions to the innumerable situations that can arise in the real world is very dangerous when applied to potentially lethal robots, and also makes the assumption of responsibility by human operators unrealistic. It is therefore apparent that the abstract reasoning of general AI approaches is inappropriate at the highest level of robot mission definition and control.

Algorithms cannot replace human responsibility. Even so, a fully testable technology such as that provided by the MEA formalism, allows for assignment of human accountability when directing robot mission outcomes and alternatives. It is possible that ever-emerging AI techniques may someday provide good methods for achieving specific individual Tactical Level behavior modules. Such employment of AI capabilities (even when experimental) can be considered appropriate in these cases since success, failure, and constraint violation remain fully accounted for by the Strategic Level MEA.

VII. Ethical Validation of Mission Definitions

A. Description Logics (DL) and a Robot Mission Ontology

Thus far, the discussion of MEA mathematical underpinnings, capabilities, and implementations has focused on providing robot operators the ability to rigorously define and test Strategic Level missions to ensure high-level mission-flow understanding sufficient for the assignment of accountability for vehicle conduct throughout the mission. The ability of an actual target vehicle to execute missions defined in this manner without further translation into a vehicle-specific form, however, has not been addressed. Mathematical logic provides a mechanism for bridging Strategic Level missions described here and vehicle-specific code for specifying and ordering Tactical Level behaviors. If properly implemented, formal logic can mathematically enforce MEA semantics in the definition of missions and during execution of those missions on target vehicles.

Description Logics (DL) are a mathematical family of logic-based knowledge representation systems that are used to describe concepts and roles within a knowledge-based system through a set of well-defined operations. DL ontologies can be used to describe the requirements and relationships of a system in a semantically meaningful way. That is, they define not only what the relationships are, but how they operate, how they are to be used, and to what specific entities they apply. As depicted in the "Ontology Spectrum" of Figure 17, DLs provide expressive power almost equal to that of First Order Logic (FOL). Further, these language constructs have been carefully defined to enable (and indeed guarantee) computationally efficient reasoning that can always identify the existence of hidden relationships and errors in the form of rule violations or contradictions (Ortiz and Šimkus 2012). These are strong capabilities with great potential value.

DLs provide the mathematical foundation of what has come to be known as the Semantic Web, an extension of the World Wide Web (Berners-Lee, Hendler and Lassila 2001). The growth of the Semantic Web has fostered the development of tools and standards that take advantage of DL logical expressiveness and mathematical rigor to provide extensive knowledge representation, discovery, and utilization capabilities. Most notably, the Web Ontology Language (OWL) (World Wide Web Consortium 2013) together with the Resource Description Framework (RDF) (World Wide Web Consortium 2014) encode a particularly powerful DL in a plain-text, XML-based, computer-readable form (Horrocks 2008). Because of its formal and general DL implementation, OWL is potentially useful beyond the Semantic Web domain. In particular, it is used here to define a robot mission description and execution ontology that applies and enforces MEA semantics. Further references of interest include (Daconta, Orbst and Smith 2003) and (Davis 2014)

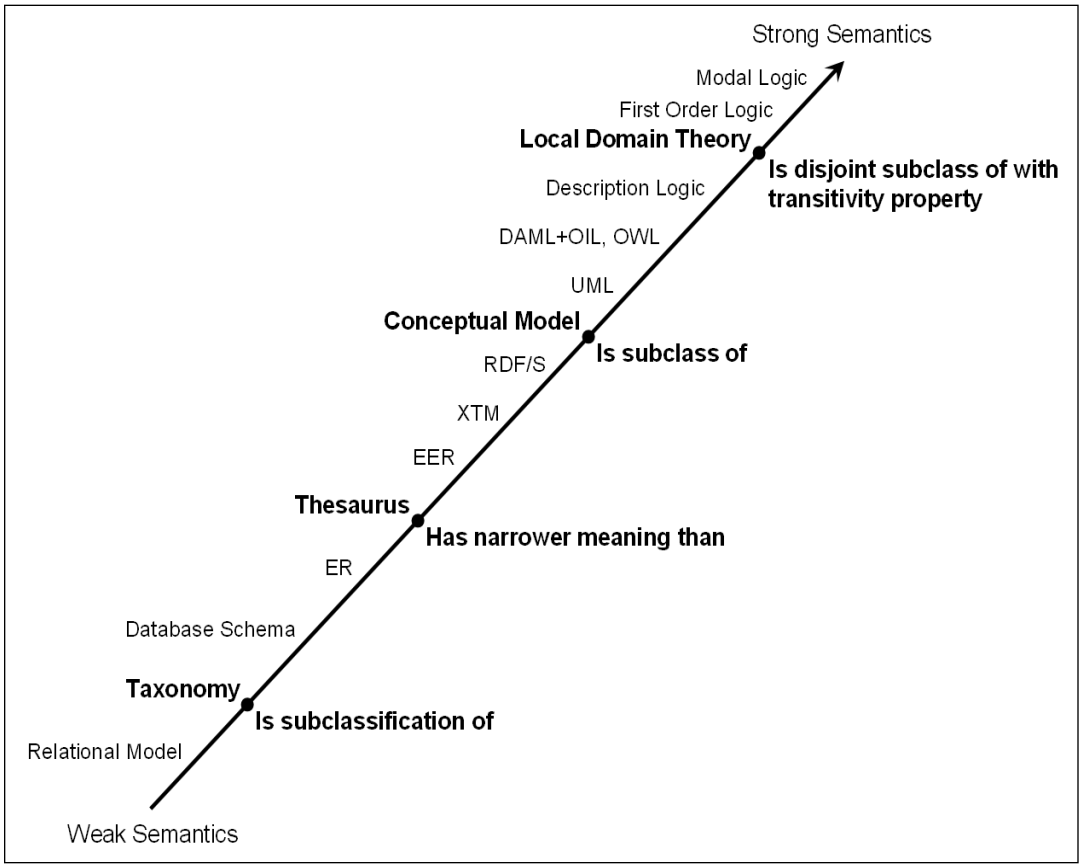

Figure 17. An "Ontology Spectrum" ranking knowledge-representation systems according to their ability to express semantic information (Daconta, Orbst and Smith 2003).

B. Mission Execution Ontology (MEO)

The mission execution ontology serves a number of purposes. First, it provides a formal and semantically rich description of the characteristics of a MEA mission description. For instance, OWL expressions are used to declare the existence of concepts such as "Mission", "Goal", and "Constraint". OWL statements are also used to define possible relationships (roles) between concepts. An entity to which the "Mission" concept applies, for example, can have an "includes" relationship with an entity to which the "Goal" concept applies. Additional OWL statements describe rules that govern how relationships are applied. As an example, a "Mission" entity must have an "includes" relationship with at least one "Goal" entity and must have a "startsWith" relationship with exactly one of those entities. A graphical depiction of the concepts and relationships defined in the mission execution ontology is provided in Figure 17. As the diagram indicates, concepts and relationships are defined to accurately represent the semantics of the previously discussed flow diagrams to include the definition of individual mission goals and constraints; goal successors in the event of goal success, failure, or constraint termination; the mission's first goal; and the application of constraints to individual goals or to the entire mission.

In addition to the "Mission", "Goal", and "Constraint" concepts that are abstracted directly from Strategic Level mission-flow diagram semantics, the mission execution ontology introduces the "Vehicle" concept. This concept provides the ability to include specific target vehicles in the mission-planning process. In particular, the "canExecute" and "canIdentify" relationships allow mission planners to explicitly assert that the intended target vehicle has a Tactical Level behavior capable of completing a particular goal and recognizing potential violation of a particular constraint, respectively. Evidently, if a mission includes goals for which the "canExecute" relationship does not exist with the intended vehicle or constraints for which the "canIdentify" relationship

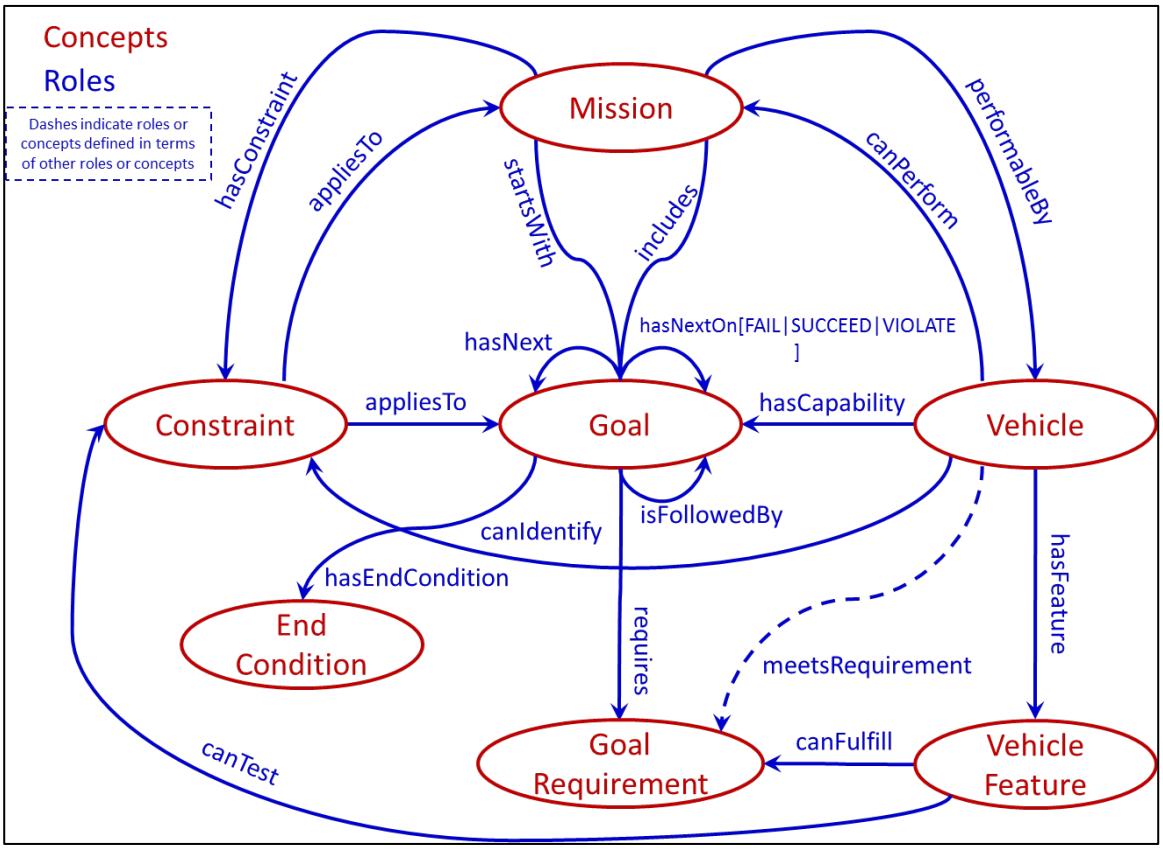

Figure 18. A Mission Execution Ontology (MEO) expressing concepts and roles (relationships across concepts) representing the flow-diagram MEA mission descriptions.

does not exist, then that mission is not appropriate for that particular vehicle. Given this requirement (which is enforced by rules within the ontology), it is impossible to define a valid mission that cannot be executed by the intended vehicle.

A second important characteristic of a DL-defined ontology is that it not only describes the rules and relationships of a knowledge domain, but also applies those rules and relationships to entities within that domain. Stated differently, the mission execution ontology does more than describe what the "Mission", "Goal", "Constraint", and "Vehicle" concepts are and how they relate to one another. It also allows the application of those concepts to real-world entities and the establishment of relationships among those entities. From a practical standpoint, this means that the atomic entities to which the "Goal" and "Constraint" concepts are applied become actual executable specifications for a set of target vehicles.

OWL provides for the incorporation of atomic entities into an ontology using Uniform Resource Identifier (URI) labels that uniquely identify individual entities. Thus, the mission execution ontology can be applied to the XML snippet above by defining an OWL statement declaring its existence and corresponding identifier. OWL statements are also used to declaratively apply concepts to and establish relationships between atomic or composite entities within the knowledge base. Figure 18 illustrates the example constraint-based mission of Figure 7 expressed in RDF/OWL syntax and logically validatable using the MEO illustrated in Figure 17, as rendered in the Stanford Protégé ontology development tool.

The ability to provide a full description of all goals and constraints within the mission execution ontology using vehicle-executable code further strengthens the MEA construct. Specifically, not only is it impossible to define a mission for a particular vehicle without explicit "canExecute" relationships between the vehicle and all mission goals and "canIdentify" relationships between the vehicle and all mission constraints, but it is also impossible to assert these relationships without an appropriate vehicle-specific encoding of all mission goals and constraints.

Finally, automated reasoning with DL-based ontologies is an important tool for ensuring Strategic Level mission validity before conducting exhaustive verification validation and accreditation (VV&A) testing using virtual simulators and real-world operations. If, for instance, an attempt is made to finalize a mission that includes goals that are not executable by the target vehicle, an OWL/RDF reasoner can quickly identify this shortcoming using the MEO. Similarly, a reasoner can detect mission flow-graph structural errors based on ontology rules that preclude illogical loops, unreachable goals, or untasked/orphan goals without specified predecessors or successors. A reasoner can also simplify the mission definition process by detecting implicit relationships that are not explicitly or correctly specified. For example, if a particular "Goal" entity

is defined and is reachable from the mission's start goal (i.e., the one with which the containing "Mission" has a "startsWith" relationship) through a sequence of goal successes and failures, then it must be in an "includes" relationship with that particular mission whether the relationship is explicitly declared or not.

As described, a DL-based mission execution ontology defined in OWL ties the MEA semantics discussed in previous sections to actual target vehicles. The ontology ensures not only the validity of the mission structure for arbitrary Strategic Level mission flow graphs, but their executability on the particular target vehicles as well. Thus, all of the requirements originally posed for assignment of human responsibility— that the mission is defined in a mathematically rigorous and fully-understood manner, that the mission specification is equally understandable by the human operator and the target vehicle, and that the mission is comprised entirely of trusted vehicle behaviors—are fully specified in the human-approved mission orders and enforced by the strict semantics encoded in the ontology.

C. Specification and Implementation of the Ontology

The principal concepts and relationships to be represented in the mission ontology were shown earlier in Figure 18. Definitions of the concepts and roles (also known as classes and properties) in this ontology need to satisfy a number of rules, as specified in Table 1, in order to support logical inferences relating to the validity of the mission structure.

Table 1. Logical Expressions defining Concepts and Roles for the Mission Planning Ontology

Rules	**Description Logic Equations**	**Plain-language description**
M = Mission Rules		
M1	Mission ⊑ ∃startsWith.Goal ⊓ =1.startsWith	A Mission can only start with a Goal and must start with exactly one Goal
M2	Mission ⊑ ∃includes.Goal ⊓ ≥1.includes	A Mission can only include Goals and must include one of more Goals
M3	Mission ⊑ ∃hasConstraint.Constraint ⊓ ≥0.hasConstraint	A Mission can be constrained only by Constraints and can have 0 or more
M4	startsWith ⊑ includes	A Mission must include the Goal that it starts with
M5	Mission ⊑ ∃performableBy.Vehicle ⊓ ≥0.performableBy	A Mission can only be performed by a Vehicle and can be performable by 0 or more Vehicles
M6	performableBy(M,V) ⊑ ∀(hasConstraint(M,C) ∘ canIdentify(V,C))	A Mission cannot be performable by a Vehicle unless that Vehicle has the ability to identify all Constraints associated with that mission
M7	performableBy(M,V)⊑ ∀(includes(M,G) ∘ hasCapability(V,G))	A Mission cannot be performable by a Vehicle unless that Vehicle has the capability to accomplish all Goals included in that Mission
V = Vehicle Rules		
V1	Vehicle ⊑ ∃hasFeature.Vehicle_Feature ⊓ ≥0.hasFeature	The only allowable features of a Vehicle are VehicleFeature. A Vehicle can have 0 or more VehicleFeatures
V2	canPerform ≡ performableBy^{-}	performableBy and canPerform are inversely equivalent
V3	meetsRequirement ≡ hasFeature ∘ canFulfill	A Vehicle meets a GoalRequirement if and only if it has a VehicleFeature that can fullfill that GoalRequirement
V4	hasFeature ∘ canTest ⊑ canIdentify	If a Vehicle has a VehicleFeature that can test a Constraint, then that Vehicle can identify that constraint
V5	hasCapability(V,G) ⊑ ∀(requires(G,R) ⊓ meetsRequirement(V,R))	If a Vehicle meets all GoalRequirements for a specific Goal, then that vehicle has the capability for that Goal

F = Feature Rules		
F1	VehicleFeature ⊑ ∃canFulfill.GoalRequirement ⊓ ≥0.canFulfill	A VehicleFeature can only fulfill GoalRequirements and may be able to fulfill 0 or more GoalRequirements
F2	VehicleFeature ⊑ ∃can_test.Constraint ⊓ ≥0.can_test	A VehicleFeature can only test Constraints and may be able to test 0 or more Constraints
C = Constraint Rules		
C1	Constraint ⊑ ∃appliesTo.(Mission ⊔ Goal)	A Constraint can apply to a Mission or a Goal (and nothing else)
C2	Constraint ⊑ ≥1.appliesTo.Goal	A Constraint must apply to at least one Goal
C3	appliesTo ∘ includes ⊑ appliesTo	A Constraint that applies to a Mission must also apply to all of the Goals that Mission includes
EC = End Condition Rules		
EC1	EndCondition ≡ {SUCCEED, FAIL, VIOLATE}	Possible ending conditions are "Succeed", "Fail", and "Violate" (i.e., imminent Constraint violation)
G = Goal Rules		
G1	Goal ⊑ ∃requires.GoalRequirement ⊓ ≥0.requires	A Goal can only require a GoalRequirement and may require 0 or more Goal Requirements
G2	Goal ⊑ ∃hasEndCondition.EndCondition ⊓ ≤1.hasEndCondition.End_Condition	A Goal's ending state must be an EndCondition, and a Goal can end with at most one EndCondition
G3	Goal ⊑ ∃isNext.Goal	A Goal can only have other Goals next
G4	hasEndCondition(G,SUCCEED) ⊔ hasEndCondition (G,FAIL) ⊔ hasEndCondition (G,VIOLATE) ⊑ isNext(G,G2)	A Goal can only have an immediate successor based on the existence of an ending state for that Goal
G5	Goal(G) ⊑ ≤1.(is_next(G,G2) ⊓ end_state(G,SUCCEED)) ⊔ ≤1.(is_next(G,G2) ⊓ end_state(G,FAIL)) ⊔ ≤1.(is_next(G,G2) ⊓ end_state(G,VIOLATE))	A Goal can have no more than one immediate successor in the event of a specific ending state
G6	Goal ⊑ ∃follows.Goal	A Goal can only be followed by another Goal
G7	Goal(G) ⊑ ¬follows(G,G)	A Goal cannot follow itself (no loops)
G8	isNext ⊑ follows	A Goal follows another goal if it is the next Goal
G9	follows ∘ follows ⊑ follows	follows is transitive (if follows(A,B) and follows(B,C), then follows(A,C))
G10	includes ≡ startsWith ∘ follows	All Goals in a Mission must **potentially** follow the starting Goal (satisfiability vice entailment)

To create a practical implementation of this ontology, we chose to employ the Protégé ontology specification tool developed by Stanford University (http://protege.stanford.edu). In the Protégé graphical user interface (GUI), classes corresponding to the concepts are shown in the left-hand panel in the screenshot in Figure 20. Annotations in the class and property definitions in Protégé reflect the plain language descriptions from Table 1 above.

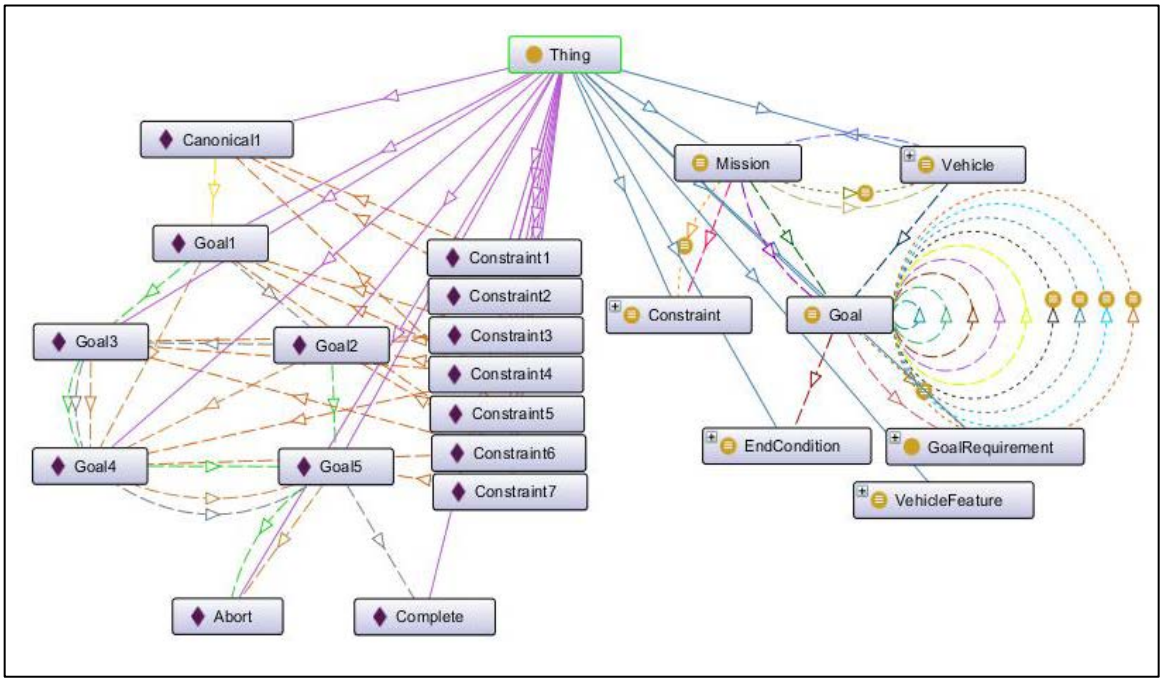

Figure 19. Validatable RDF/OWL diagram of goals, relationships, assertions, and ethical constraints for the canonical mission of Figure 7. Rules and rendering produced using Protégé Ontology Editor (Stanford University 2016).

In this example, the `Mission` class is defined as equivalent to the class of all `Things` that has some `Constraint` (zero or more), starts with exactly one `Goal`, includes a minimum of one `Goal`, and is performable by some `Vehicle` (one or more). As a software application constructs instances for the knowledge base, it can focus on the properties and allow an automated reasoner to determine if the information correctly and completely specifies a mission; i.e., to classify a `Thing` having various property assertions as a `Mission` meeting the class specification. This approach provides a direct way to check the logical consistency of the created instances, rather than explicitly creating individuals of the `Mission` class (even without any properties, which remains valid under the open world assumption that something cannot be declared false just because it is not known to be true).

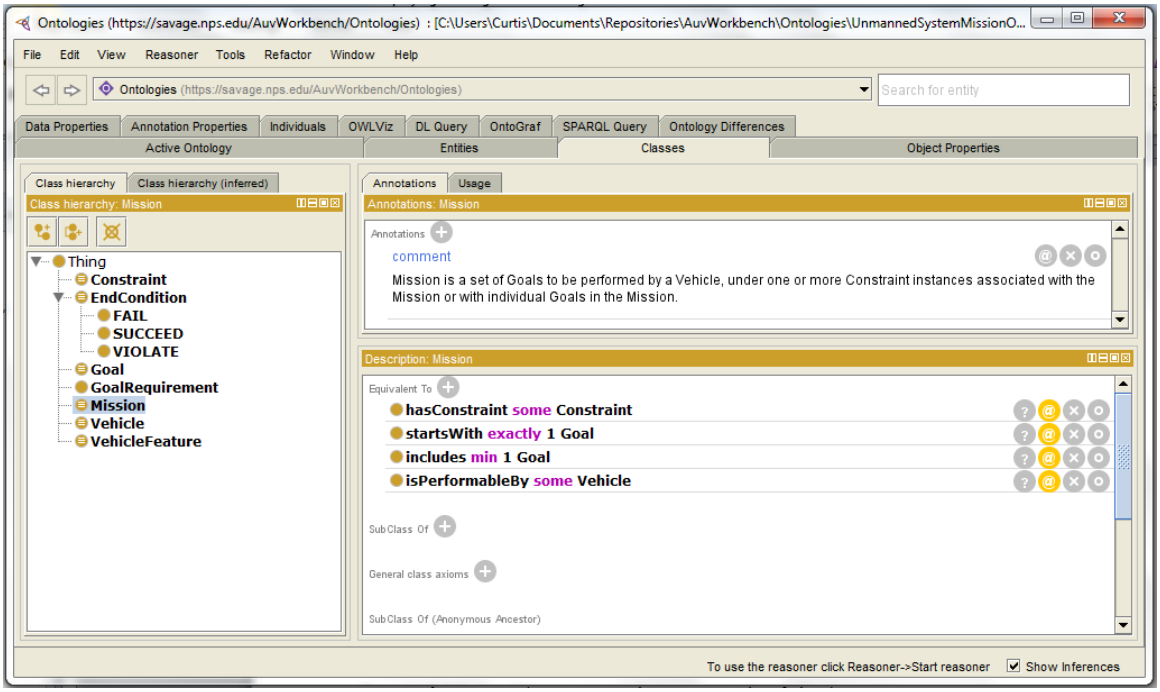

Figure 20. Protégé graphical user interface (GUI) showing the class hierarchy on left and specification of Mission class in lower right.

Some of the rules in Table 1 are captured in the class hierarchy. For example, the disjoint covering of `EndCondition` by subclasses `FAIL`, `SUCCEED`, and `VIOLATE` (Rule EC1) is represented by defining class `EndCondition` as equivalent to the union of these subclasses, and a declaration that the subclasses are mutually disjoint. This approach is called a *value partition* in the literature (Horridge 2011). Defining these `EndCondition` "values" as subclasses (rather than, say, individuals) facilitates extension of the ontology through addition of other subclasses of end conditions and through possible subcategorization of each end condition, such as different `FAIL` conditions that later might need to become explicit in the ontology. This method also supports useful logical operations in classifying mission goal transitions and actual mission performance according to the assigned `EndConditions`.

The specification of object properties encodes several of the rules from Table 1 above. A screenshot showing the hierarchy of object properties, with details on the definition of the `includes` property, is shown in Figure 21.

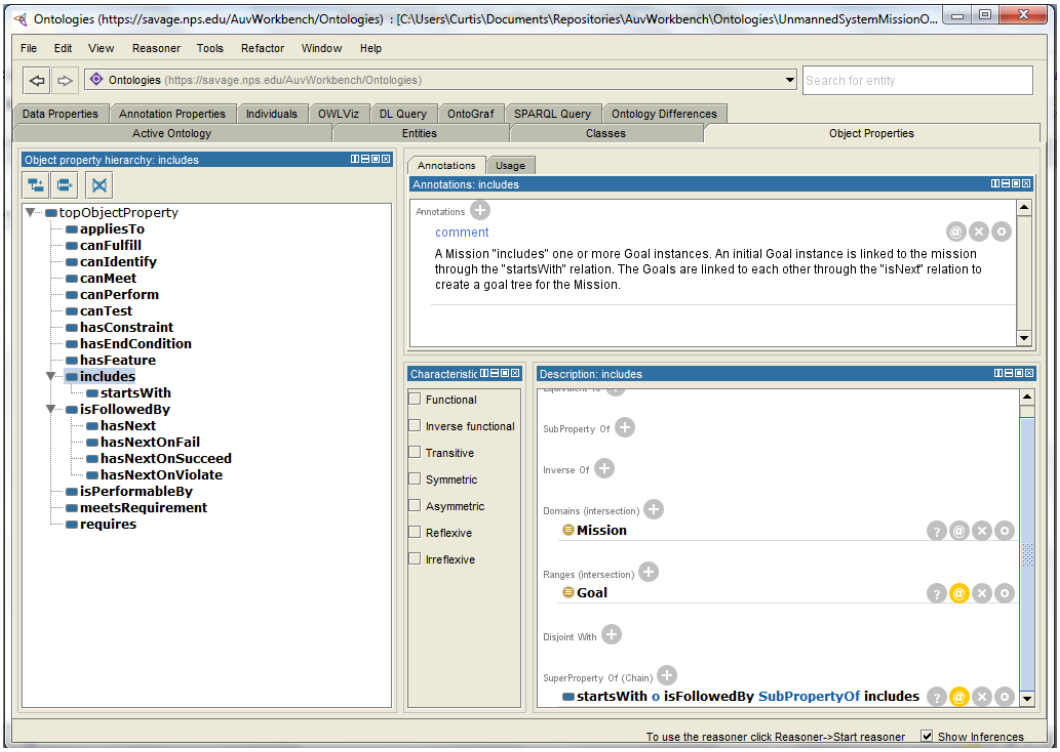

Figure 21. Object property hierarchy in Protégé and specification of the `includes` property.

Several of the rules in Table 1 are addressed by the assertion of Domain and Range constraints on the properties. In the definition of the `includes` property above, a `Mission` (domain) can only include `Goals` (range). This represents part of rule M2. The other part of rule M2 (that a `Mission` must include at least one `Goal`) is represented in the assertion `includes min 1 Goal` shown previously in the definition of the Mission class in Figure 19. In the same way, domain and range constraints on the property definitions partially address rules M1, M2, M3, M5, V1, F1, F2, and C1 (with the class definitions addressing the other parts of those rules dealing with cardinality on the properties, as shown in the definition of the `Mission` class above). Note that when there is exactly one association allowed on a property, it is asserted both in the class definition on the cardinality (e.g., in the definition of the Mission class: `startsWith exactly 1 Goal`) and in the property definition (*Functional* characteristic is checked in the Object Property window in Protégé).

In the Object Property description, the property can be identified as Functional, Inverse Functional, Transitive, Symmetric, Asymmetric, Reflexive, and Irreflexive. These characteristics address rules in Table 1 such as M1 (`startsWith` is functional) and G9 (`follows` is transitive), as well as providing additional specificity to such properties as `hasEndCondition` (functional) and `isNext` (irreflexive; i.e., a goal cannot come after itself). In addition, explicit assertions can be made regarding the properties, as in `canPerform` is the inverse of `performableBy` (rule V2) and `startsWith` is a subproperty of `includes` (rule M4).

Definition of the `includes` property in Figure 21 provides an example of the use of a "property chain" in the definition. The definition states that includes is a superproperty of the property chain formed by the composition of `startsWith` and `isFollowedBy`, addressing one direction of the logical equivalence stated in Rule G10 (note: in Protégé, there is no way to represent the assertion that `includes` is a subproperty of the property chain, in order to obtain the logical equivalence, other than by creating a new property that also has the above superproperty assertion, and then stating in the definition of

`includes` that it is both a subproperty of the newly defined property and a superproperty of the property chain as shown). This method is used to address rules V3 (`meetsRequirement` is a superproperty on the composition of `hasFeature` and `canFulfill`), V4 (`canIdentify` is a superproperty on the composition of `hasFeature` and `canTest`), and C3 (`appliesTo` is a `superproperty` on the composition of `appliesTo` and `includes`).

There are difficulties in OWL 2 in expressing the universal quantifier in Rules M6, M7, and V5. For example, Rule M6 states that a `Mission` is performable by a `Vehicle` if that `Vehicle` has the ability to identify all `Constraints` associated with that `Mission`. It is straightforward to express the assertion from the opposite perspective; that is, if a `Mission` is `performableBy` a `Vehicle`, then the `Vehicle` has the ability to identify (`canIdentify`) all the `Constraints` associated with that `Mission`. Given the former, the reasoner could infer the latter. Note that the negation of each side of Rule M6 can be expressed with existential quantifiers; as in: a `Mission` is **not** `performableBy` a `Vehicle` if there exists a `Constraint` that **cannot** be identified (negation of `canIdentify`) by the `Vehicle`. Similarly, Rule M7 states that a `Mission` cannot be performable by a `Vehicle` unless that `Vehicle` has the capability to accomplish all `Goals` included in that `Mission`. Finally, Rule V5 states that if a `Vehicle` meets all `GoalRequirements` for a specific `Goal`, then that `Vehicle` `canMeet` (has the capability to achieve) that `Goal`. Implementation here involves ongoing work in the use of OWL and expression of rules, an area of further study for purposes of the MEO (Krisnadhi, Maier and Hitzler 2011).

Regarding Rule G4 (a `Goal` can only have an immediate successor based on the existence of an ending state for that `Goal`) and Rule G5 (a `Goal` can have no more than one immediate successor in the event of a specific ending state), the approach taken in the ontology is to have explicit `isNextOn`* relations (where * is `Fail`, `Succeed`, or `Violate`) to allow specification of the branching to other goals based on the end condition. During mission planning, these relations express the link between a goal and successive goals based on the end condition; i.e., what goal follows on a `Fail` condition, what goal follows on a `Succeed` condition, and what goal follows on a `Violate` condition. During mission execution, when an actual end condition occurs during performance of a goal, the end condition would be recorded in the `hasEndCondition` and the actual goal selected to follow based on that condition would be recorded in the `isNext` relation (i.e., these become mission log entries rather than part of the initial plan description).

Finally, regarding Rule G7 stating a goal cannot follow itself (no loops at strategic level), there is no abstract way to express this rule in the ontology per se (i.e., in the T-Box assertions), since enforcement of the rule requires logic dealing with a specific individual rather than generic members of the class. An executable implementation can readily perform this check in software operating in conjunction with the reasoner. In addition, a rule can be added to infer that a constraint that applies to a goal also applies to the mission that contains that goal (the ontology already has the converse, where a constraint on a mission applies to all the goals in the mission) simply by making `hasConstraint` a subproperty of the property chain `includes o appliesTo`$^{-1}$.

Figure 22 shows the classes and relationships as displayed by the Protégé OntoGraf plug-in. The key for the color-codings of the relationships is provided in Protégé in the OntoGraf tool. A partial display of the key is shown in Figure 23. Exploration of ontological relationships is helpful for illustrating correctness, completeness and logical consistency.

Several examples of mission specifications and their ontological representations are available from the authors.

D. Mission Execution Ontology Summary

It is possible to produce general robot mission orders that are understandable by (legally culpable) humans and are reliably and safely executable by robots. The semantic representation of the mission plans permits automated examination of the plans for logical consistency and provides an enhanced methodology for software implementations to process missions. Even if perfectly executable, proper robot logic is not useful in military context unless it is a directly compatible extension of warfighter logic. ROEs, concepts of operation, doctrine, tactics, etc. must be expressible in equivalent terms to be effective and usable. Constraint tests must be determinable by a human supervisor or critic, by a virtual environment running a simulation, or by on-board robot sensors in the operating environment. Constraint test can match common guidelines such as rules of the road, water-space management, ROEs, operational orders, and other expressions of bounds on mission conduct. These expressions cannot be vague, must result in clear logical determinism (true or false), must be able to combine multiple logical constraints, and need to note reporting requirements when human permission is necessary. For strategic-level task controllers, the ternary tactical task sequencer using ethics constraints may allow traceability and accountability for the full set of executed robot tasks without loss of generality. ROEs and other expressions provide ethical constraints and boundary conditions on robot strategic planning and operational conduct that can work cooperatively and satisfactorily with humans.

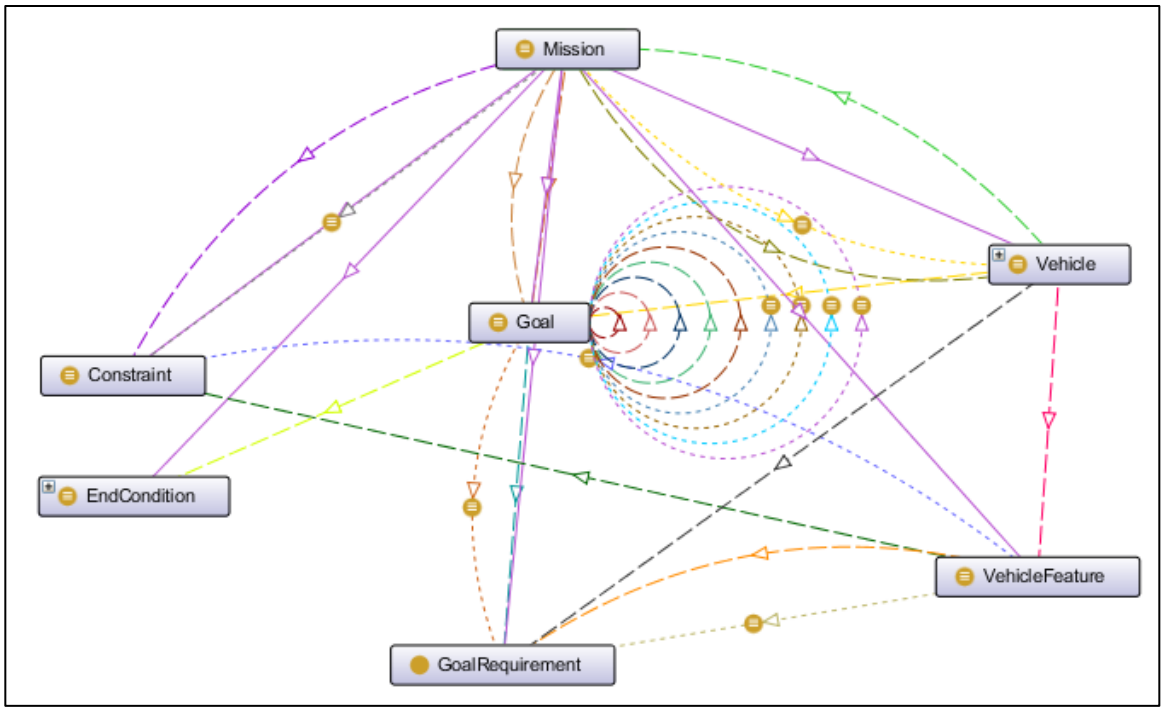

Figure 22. OntoGraf representation of the Mission structure in Protégé (essentially, the right-hand side of Figure 18 in a slightly different layout to better illustrate ontology relationships).

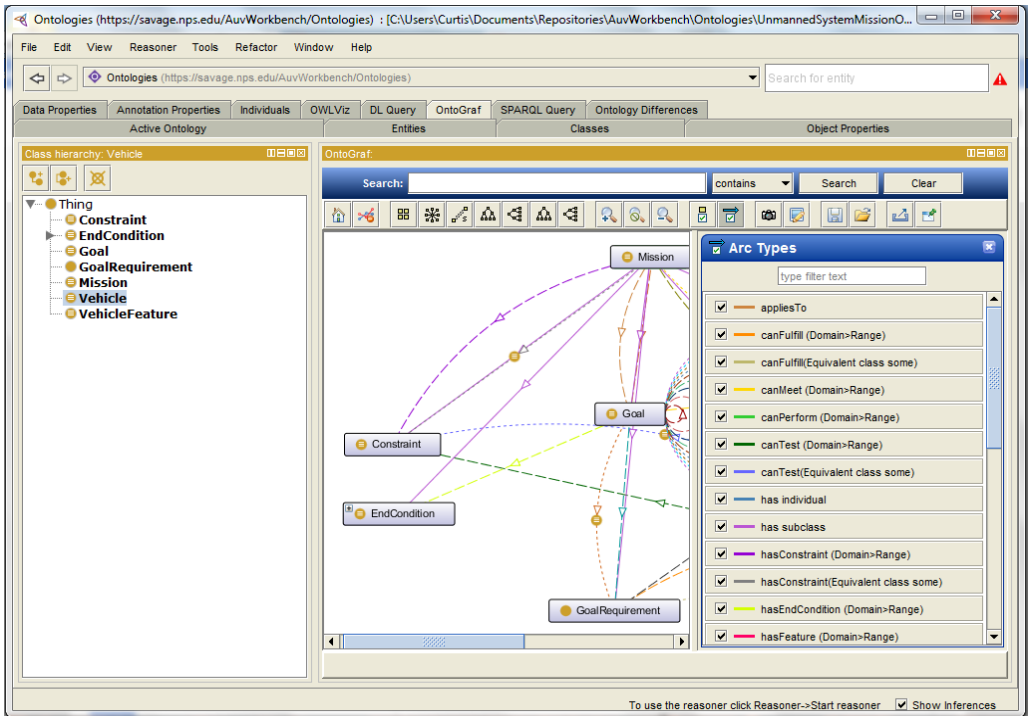

Figure 23. OntoGraf color-coding key associating property names to links in the displayed graph.

VIII. Findings, Conclusions, and Recommendations

Humans given authority over potentially lethal robotic systems must be provided with realistic capabilities that enable meaningful supervision, responsibility and accountability. Autonomous Vehicle Command Language (AVCL) mission definitions provide one such example: the ability to define Strategic Level goals and Tactical Level tasks in a human-understandable way, and in syntactically validatable form, that a wide variety of robots might interpret and execute. Mission Execution Automata (MEA) formalisms show that the underlying programming constructs are tractable and sufficiently general to ensure broad feasibility. Mission Execution Ontology (MEO) validation provides further abilities to logically evaluate the semantic correctness and completeness of ethically constrained mission definitions. Together these capabilities provide a practical framework for ethically grounded human supervision of unmanned systems.

Key findings of this work are summarized below:

- An MEA is a generalization of a TM in the sense that it has communication links to at least one external agent that may be human or mechanical (or both). External agents have predefined input and output symbols/messages.
- A *fully testable* MEA has no loops in its state graph, and does not access its tape.
- We recommend that the top level of RBM code consist of a single fully testable MEA. Other levels of RBM can use MEAs without this restriction.
- MEOs can be employed before mission execution to make sure that a given mission for a specified vehicle will execute and will terminate within a prescribed time.

The research findings lead to the following general conclusions and recommendations for future work regarding unmanned systems that have the capacity for lethal force.

- *Ethical operation of autonomous systems requires human responsibility, accountability and understanding.* Any decision to deploy potentially lethal force without appropriate constraints or control may be dangerous, immoral and illegal.
- *Lethality requires an ethical and legal basis for unmanned system operations*. Naval and military applications have well-defined requirements that are ultimately guided by ethical requirements such as the Law of Armed Conflict (LOAC) and treaty-based statutes. Similar concerns and aspects exist in civil robotics (e.g. safety of self-driving cars). Principles such as vicarious liability clearly show that humans are both responsible and also vulnerable. Robot self-preservation becomes irrelevant when human life is at stake.
- *Unmanned systems can remain supervised and semi-autonomous, even if communications are lost, if appropriate guidance and checkpoints are provided.* Robotic systems are capable of following well-defined orders within measurable constraints. For today's systems, loss of control typically occurs when shifting from direct supervision to semi-autonomous control to independent operations. Mission orders need to account for such circumstances correctly, or a system should not be deployed.
- *Human clarity and cooperative action are essential for supervising robots together with human teams.* No magical on-board ethics-agent homunculus will be developed that substitutes for careful human planning and direction. Meanwhile, even a perfect robot is unethical if not understood by humans. Robots today are not capable of refusing to obey an order through complex inferencing from general principles. Thus robot tasking must have explicit constraints to avoid misapplication of potentially lethal force.
- *Unmanned systems can be compatibly tasked in concert with human teams.* Ethical control of lethality is achieved via well-defined missions and well-understood constraints. "Approved robot software that includes everything needed" is analogous to "the military operator is fully trained and qualified." International military teams provide existence proofs on a daily basis that collaborative approaches are feasible, despite differences in operational policies, procedures and language. This work presents a corresponding approach which historically and currently works well for formal tasking of human teams.
- *Applied ethics equals defining tasks and observing constraints prior to executing potentially harmful tasks*. No human operator or team undertakes potentially life-threatening activities without knowing necessary precautions and following the rules. Tasking must therefore be clear, and prerequisite constraints must match necessary legal, safety and policy requirements.
- *The mathematical concepts of description logics and ontologies, as implemented by Semantic Web technologies, is proposed to capture common logical and ethical relationships for mission and task definition*. This approach to mission validation is mathematically rigorous and formally well-defined. Mission correctness becomes fully validatable using widely deployed technology. The defined relations and rules capture broad common practices for mission planning of robot and military missions. A constraint-based approach to mission definition appears to present fundamental value.
- *A mission-definition approach to constrained tasking is actionable for all unmanned systems regardless of software architecture*. The authors have carefully compared these mission-definition mechanisms to a wide variety of human-control approaches in the real world, in particular focusing on cooperative maritime and naval operations. Although primarily in the maritime domain, a wide variety of unmanned systems have also been compared over the years. This evaluation leads to specific conclusions that
  - The approach is practical, repeatable, and thorough.
  - The approach is compatible with a variety of robot system architectures.
  - Human role remains essential throughout, even when directness of control varies for remotely operated systems.

- *For those choosing to adopt RBM for robot control, we urge that the strategic level be fail-safe and exhaustively testable.* This means that this level can be based only on finite state machine theory and propositional logic, and not predicate logic, since systems based on the latter are generally undecidable.
- *Ethical constraints on robot mission execution are possible today.* There is no need to wait for notional future developments in Artificial Intelligence (AI). It is moral imperative that ethical constraints in some form be introduced into the software of all robots capable of inflicting unintended harm to humans or property.

Ethical operation of robotic systems requires human accountability. In both the legal and moral sense, this implies that human operators be in a position to understand, and therefore control, robot mission outcomes. This level of understanding can be achieved through the satisfaction of three requirements: operator understanding of high-level mission flow, mission descriptions understandable to both human operators and target vehicles, and mission descriptions consisting entirely of trusted behaviors and constraints.

Early NPS UUV missions were executed as a result of an inferencing process over a mission definition comprised of a set of rules and facts. This approach provided no means of proving the correctness of strategic-level missions comparable to the exhaustive testing of MEA flow graphs. For this type of system, errors in the mission axiom set can lead to unpredictable and potentially hazardous or self-defeating system execution behavior. Ultimately, this unpredictability precludes the formal assumption of responsibility or liability for robot missions.

AI approaches in general almost invariably make use of easily confounded inferential reasoning or statistical pattern recognition. Applying such broad mathematical abstractions to the innumerable situations that can arise in the real world is inherently unpredictable, and also makes unrealistic any assumption of responsibility by human operators. It is therefore apparent that the abstract reasoning of general AI approaches is inappropriate, at least at the present time, for the highest level of robot mission definition and control.

Algorithms cannot replace human responsibility. Even so, a fully testable technology (such as that provided by the MEA and MEO formalisms) allows for the assignment of human accountability. Specifically, the MEA provides a mathematically rigorous mechanism for mission definition and execution as an exhaustively testable flow diagram. This approach ensures that accountable operators can fully understand all high-level task sequences before authorizing robot operations. The MEO employs DLs and Semantic Web technologies to provide strong assurances that MEA mission definitions are semantically correct and fully executable by specific target vehicles.

By applying the best strengths of human ethical responsibility, repeatable formal logic and directable unmanned systems together, these capabilities provide a practical framework for ethically grounded human supervision of unmanned systems. Much important work awaits.

ACKNOWLEDGMENT

The authors gratefully acknowledge the fundamentally important insights and continuing contributions of many dozens of NPS students and colleagues. The content of this paper reflects the opinions of the authors and not necessarily the position of the Naval Postgraduate School or the Department of the Navy.

REFERENCES

Albus, J. "Engineering Intelligent Systems." *Proceedings of the IEEE ISIC/CIRA/ISAS Joint Conference*. Gaithersburg, MD, 1998.

Arkin, R. *Governing Lethal Behavior in Autonomous Robots.* Boca Raton, FL: Taylor & Francis Group, 2009.

Azimov, I. *I, Robot (2004 ed.).* New York, NY: Bantam Dell, 1950.

Berners-Lee, T, J Hendler, and O Lassila. "The Semantic Web." *Scientific American*, May 2001: 34-43.

Brtuzman, D. *A Virtual World for an Autonomous Underwater Vehicle.* Ph.D. Dissertation, Monterey California USA: Naval Postgraduate School, 1994.

Brutzman et al. *Autonomous Unmanned Vehicle (AUV) Workbench.* September 18, 2016. https://savage.nps.edu/AuvWorkbench/ (accessed December 9, 2016).

—. "Autonomous Vehicle Commandl Language." January 1, 2013. https://savage.nps.edu/Savage/AuvWorkbench/AVCL/AVCL.html (accessed December 9, 2016).

Brutzman, D. P., D. T. Davis, G. R. Lucas, Jr., and R. B. McGhee. "Run-Time Ethics Checking for Autonomous Unmanned Vehicles: Developing a Practical Approach." *Proceedings of the 18th International Symposium on Unmanned Untethered Submersible Technology.* Portsmouth, NH, 2013.

Brutzman, D. P., R. B. McGhee, and D. T. Davis. "An Implemented Universal Mission Controller with Run Time Ethics Checking for Autonomous Unmanned Vehicles--a UUV Example." *Proceedings of the OES-IEEE Autonomous Underwater Vehicles 2012.* Southampton, UK, 2012.

Brutzman, Don, Tony Healey, Dave Marco, and Bob McGhee. "The Phoenix Autonomous Underwater Vehicle." In *AI-Based Mobile Robots.* Cambridge: MIT/AAAI Press, 1998.

Byrnes, R. B., A. J. Healey, R. B. McGhee, M. L. Nelson, S. Kwak, and D. P. Brutzman. "The Rational Behavior Software Architecture for Intelligent Ships." *Naval Engineers Journal*, March 1996: 43-56.

Byrnes, R. *The Rational Behavior Model: A Multi-Paradigm, Tri-Level Software Architecture for the Control of Autonomous Vehicles.* Ph.D. Dissertation, Computer Science, Naval Postgraduate School, Monterey, CA: Naval Postgraduate School, 1993.

Čapek, K. *Rossum's Universal Robots (2004 ed.).* Translated by Claudia Novack. New York, NY: Penguin Group, 1921.

Daconta, M D, L J Orbst, and K T Smith. *The Semantic Web, A Guide to the Future of XML, Web Services, and Knowledge Management.* Indianapolis, IN: Wiley Publishing, 2003.

Dannegger, C. "Real-Time Autonomic Automation." In *Springer Handbook of Automation*, edited by Shimon Y. Nof, 381-404. New York, NY: Springer, 2009.

Davis, D. T. *Design, Implementation, and Testing of a Common Data Model Supporting Autonomous Vehicle Compatibility and Interoperability.* Ph.D. Dissertation, Computer Science, Naval Postgraduate School, Monterey, CA: Naval Postgraduate School, 2006.

Davis, Duane T. *Semantic Web and Inferencing Technologies for Department of Defense Systems.* Technical Report, Center for Multi-INT Studies, Naval Postgraduate School, Monterey, CA: Naval Postgraduate School, 2014.

Department of Defense. "Autonomy in Weapon Systems, Directive 3000.09." November 21, 2012. http://www.dtic.mil/whs/directives/corres/pdf/300009p.pdf (accessed December 23, 2014).

Duarte, C. N., et al. "A Common Control Language to Support Multiple Cooperating UAVs." *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology.* Durham, NH, 2005.

Hammond, G. T. *The Mind of War: John Boyd and American Security.* Washington, DC: Smithsonian Institution Press, 2001.

Horridge, Matthew. "A Practical Guide to Building OWL Ontologies using Protege 4 and CO-ODE Tools. Edition 1.3." March 24, 2011. http://owl.cs.manchester.ac.uk/publications/talks-and-tutorials/protg-owl-tutorial/ (accessed December 23, 2014).

Horrocks, I. "Ontologies and the Semantic Web." *Comunications of the ACM* 51, no. 12 (2008): 58-67.

Krisnadhi, Adila, Frederick Maier, and Pascal Hitzler. "OWL and Rules." In *Reasoning Web: Semantic Technologies for the Web of Data*, edited by A. Polleres, et al., 382-415. Heidelberg: Springer, 2011.

Langley, Charles, and Clive Spenser. "The Visual Development of Rule-Based Systems." *PC AI Magazine*, 2005: 29-36.

Lin, Patrick, Keith Abney, and George A. Bekey, . *Robot Ethics: The Ethical and Social Implications of Robotics.* MIT Press, 2011.

Lokhorst, G., and J. ven den Hoven. "Responsibility for Military Robots." In *Robot Ethics: The Ethical and Social Implications of Robotics*, edited by Nancy G. Lin, Keith Agney and George A. Bekey, 145-156. Cambridge, MA: MIT Press, 2012.

Mack, W. P., H. A. Seymour, and L. A. McComas. *The Naval Officer's Guide, 11th ed.* Annapolis, MD: Naval Institute Press, 1998.

Marco, D.B., A.J. Healey, and R.B. McGhee. "Autonomous Underwater Vehicles: Hybrid Control of Mission and Motion." *Autonomous Robots 3*, 1996: 169-186.

McGhee, R. B., D. P. Brutzman, and D. T. Davis. *A Taxonomy of Turing Machines and Mission Execution Automata with Lisp/Prolog Implementation.* Technical Report, MOVES Institute, Naval Postgraduate School, Monterey, CA: Naval Postgraduate School, 2011.

—. "A Universal Multiphase Mission Execution Automaton (MEA) with Prolog Implementation for Unmanned Untethered Vehicles." *Proceedings of the 17th International Symposium on Unmanned Untethered Submersible Technology.* Portsmouth, NY, 2011.

McGhee, R. B., D. P. Brutzman, and D. T. Davis. *Recursive Goal Refinement and Iterative Task Abastraction for Top-Level Control of Autonomous Mobile Robots by Mission Execution Automata--a UUV Example.* MOVES Institute, Naval Postgraduate SChool, Monterey, CA: Naval Postgraduate School, 2012.

Minsky, M. *Computation: Finite and Infinite Machines.* Englewood Cliffs, NJ: Prentice Hall, 1967.

Ortiz, M, and M Šimkus. "Reasoning and Query Answering in Description Logics." *Reasoning Web 2012, Volume 7487 of Lecture Notes in Computer Science.* Berlin Heidelberg: Springer-Verlag, 2012.

Petzold, C. *The Annotated Turing: A Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine.* Indianapolis, IN: Wiley Publishing, 2008.

Posadas, S. *Stochastic Simulation of a Commander's Decision Cycle (SSIM CODE).* Monterey: Master's Thesis, Naval Postgraduate School, 2001.

Ricard, M., and S. Kolitz. "The ADEPT Framework for Intelligent Autonomy." *Intelligent Systems for Auronautics Workshop.* Brussels, Belgium, 2002.

Rowe, N.C. *Artificial Intelligence Through Prolog.* Englewood Cliffs: Prentice Hall, 1988.

Scharre, P. "Autonomous Weapons and Operational Risk." *Ethical Autonomy Project, Center for a New American Security (CNAS).* February 2016. http://www.cnas.org/autonomous-weapons-and-operational-risk.

Simpson, Stephen G. *Logic and Mathematics, in The Examined Life: Readings from Western Philosophy from Plato to Kant.* Edited by Stanley Rosen. Random House, 2000.

Skiena, Steven S. *The Algorithm Design Manual.* 2. London: Springer-Verlag, 1997.

Sparrow, R. "Just say "No" to Drones." *Technology and Society Magazine* 31, no. 1 (2012): 56-63.

U.S. Army Training and Doctrine Command Analysis Center. "COMBATXXI." *U.S. Army TRADOC Analysis Center.* September 29, 2015. http://www.trac.army.mil/COMBATXXI.pdf (accessed January 28, 2016).

"Vicarious Liability." In *Merriam-Webster's Dictionary of Law*. Springfield: Merriam-Webster, 2011.

World Wide Web Consortium. *Resource Description Framework (RDF) Semantic Web Standards.* March 15, 2014. https://www.w3.org/2001/sw/wiki/RDF (accessed December 9, 2016).

—. *Web Ontology Language (OWL) Semantic Web Standards.* December 11, 2013. https://www.w3.org/2001/sw/wiki/OWL (accessed December 9, 2016).

Examples:

[1] IEEE Criteria for Class IE Electric Systems, IEEE Standard 308, 1969.

[2] Letter Symbols for Quantities, ANSI Standard Y10.5-1968.

Don Brutzman Don Brutzman is a computer scientist and Associate Professor of Applied Science working in the Undersea Warfare Academic Group and Information Sciences Department at the Naval Postgraduate School (NPS) in Monterey California. He is cochair of the Extensible 3D (X3D) Graphics Working Group and a founding member of the Board of Directors for the non-profit Web3D Consortium. He is lead author of the book X3D Graphics for Web Authors, published April 2007 by Morgan Kaufmann. He is a retired naval submarine officer and principal investigator for the Network-Optional Warfare (NOW) and robodata projects. His research interests include underwater robotics, real-time 3D computer graphics, artificial intelligence, and high-performance networking.

Duane Davis Duane T. Davis, Ph.D. is a Research Professor with the Naval Postgraduate School Computer Science Department and has been a member of the NPS faculty since 2008. His teaching and research interests include multi-vehicle robotic systems, swarm robotics, cybersecurity and operations, and ethical employment of autonomous and cyber capabilities. In addition to his instructional and research roles, he serves as the Academic Associate of the Cyber Academic Group and is responsible for the development and maintenance of learning objectives, educational goals, and course content for five cyber degree and certificate programs. Prior to assuming his current position, Dr. Davis was a career naval officer, serving for more than 20 years in the aviation community. While on active duty he held leadership positions in both operational and educational environments, and participated in deployments in support of numerous real-world operations.

Curtis L. Blais received the B.S. degree in mathematics and the M.S. degree in mathematics from the University of Notre Dame, South Bend, Indiana in 1972 and 1973, respectively. From 1999 to present, he has been a member of the research faculty at the Naval Postgraduate School (NPS), Monterey, California, in the Modeling, Virtual Environments, and Simulation (MOVES) Institute. Mr. Blais conducts research and teaches in modeling and simulation, encompassing simulation development processes, standards, human behavior modeling, and information modeling. Prior to NPS, Mr. Blais worked for 25 years in modeling and simulation (M&S), first as a mathematician/operations research analyst for the Space and Naval Warfare Systems Command in San Diego, California, and then as software engineer, project manager, and department manager for Systems Exploration Incorporated and Visicom. Over those years, Mr. Blais worked on numerous simulation projects, including development and delivery of several generations of command staff training systems for the United States Marine Corps from 1976 through 1999. Current research interests include improvement of human behavior models to distinguish human performance from robotic system performance in analytical combat models. Mr. Blais is a member of the IEEE, Simulation Interoperability Standards Organization (SISO), and the Society for Modeling and Simulation International (SCS).

Robert B. McGhee was born in Detroit, Michigan in 1929. He received a B.S. degree in Engineering Physics from the University of Michigan in 1952. After graduation, he served in the U.S. Army Ordnance Corps as a guided missile maintenance officer. Subsequently, beginning in 1955, he worked as a guided missile engineer for Hughes Aircraft Company, in Culver City, California, while also pursuing graduate studies at the University of Southern California. He received an M.S. degree in 1957, and a Ph.D. degree in 1963, from USC, both in electrical engineering. In 1963 he joined the faculty of Electrical Engineering at USC, first as an Assistant Professor and subsequently as an Associate Professor. In 1968, he accepted a position of Professor of Electrical Engineering at Ohio State University, in Columbus, Ohio, where he remained for 17 years. In 1985, he joined the Naval Postgraduate School as Professor of Computer Science until 2004 when he retired and was appointed Professor Emeritus.

From the beginning of his career, his research activities have, centered on various aspects computer engineering and robotics, where he has remained active up to the present time. For the past six years he has been primarily concerned with developing mathematical concepts and practical software architectures capable of providing military robots with a capacity to observe ethical constraints on their behavior during mission execution, and able to refuse to execute illegal orders. He is a Fellow of the IEEE in the area of Robotics.