Proceedings of the 2005 Knowledge Intensive Multi-Agent Systems (KIMAS) Conference,
Waltham, MA, April 18-21, 2005.

# An Intelligent Ground Vehicle Ontology for Multi-Agent System Integration

Craig Schlenoff, National Institute of Standards and Technology, Gaithersburg, MD, craig.schlenoff@nist.gov
Randy Washington, DCS Corporation, Alexandria, VA, rwashington@dcscorp.com
Tony Barbera, National Institute of Standards and Technology, Gaithersburg, MD, tony.barbera@nist.gov

**Abstract** — The level of automation in ground combat vehicles being developed for the Army's objective force is greatly increased over the Army's legacy force. The development of these intelligent ground vehicles (IGV) requires a thorough understanding of all of the intelligent behavior that needs to be exhibited by the system so that designers can allocate functionality to humans and/or machines. In this paper, we describe the joint effort currently being performed by DCS Corporation and NIST to develop an intelligent ground vehicle (IGV) ontology using Protégé. The goal of this effort is to develop a common, implementation-independent, extendable knowledge source for researchers and developers in the intelligent vehicle community. This paper describes the methodology we have used to identify knowledge in this domain and an approach to capture and visualize the knowledge in the ontology.

## 1. INTRODUCTION

The level of automation in ground combat vehicles being developed for the Army's objective force is greatly increased over the Army's legacy force. This automation is taking many forms in emerging ground vehicles; varying from operator decision aides to fully autonomous unmanned systems. The development of these intelligent ground vehicles (IGV) requires a thorough understanding of all of the intelligent behavior that needs to be exhibited by the system so that designers can allocate functionality to humans and/or machines. Tradition system specification techniques focused heavily on the functional description of the major systems of a vehicle and implicitly assumed that a well-trained crew would operate these systems in a manner to accomplish the tactical mission assigned to the vehicle. In order to allocate some or all of these intelligent behaviors to machines in future ground vehicles, it is necessary to be able to identify and describe these intelligent behaviors.

The U.S. Army Tank Automotive Research, Development and Engineering Center (TARDEC) has funded DCS Corporation and the National Institute of Standards and Technology (NIST) to explore approaches to model the ground vehicle domain with explicit representation of intelligent behavior. This exploration has included the analysis of modeling languages (i.e., UML, DAML, OWL) as well as reference architectures. A major component of this effort has been the development of an IGV Ontology.

NIST and DCS Corporation have taken the view that an IGV can be viewed as a multi-agent system, where agents can represent components within the vehicle (e.g., a propulsion system, a lethality system, etc). In addition, an IGV, as a whole, can serve as a single agent within a troop, platoon, or section, where multiple IGVs are present. In order for a group of agents to work together to accomplish a common goal, they must be able to clearly and unambiguously communicate with each other without the fear of loss of information or misinterpretation. We have used the IGV Ontology to specify a common lexicon and semantics to address this challenge.

In this paper, we describe the joint effort currently being performed by DCS Corporation and NIST to develop an intelligent ground vehicle (IGV) ontology using Protégé. The goal of this effort is to develop a common, implementation-independent, extendable knowledge source for researchers and developers in the intelligent vehicle community that will:

- Provide a standard set of domain concepts along with their attributes and inter-relations;
- Allow for knowledge capture and reuse;
- Facilitate systems specification, design, and integration, and;
- Accelerate research in the field.

This paper describes the methodology we have used to identify knowledge in this domain and an approach we have used to capture and visualize the knowledge in an ontology. Section 2 describes the Real-time Control System (RCS) and its underlying methodology that we use to determine the information requirements to model in the ontology. Section 3 describes the IGV Ontology. Section 4 describes our current status. Section 5 concludes the paper.

## 2. THE REAL-TIME CONTROL SYSTEM (RCS)

### 2.1. The Methodology

One of the first steps in any ontology development effort is to identify the information requirements that are necessary to be modeled in the ontology. For this effort, we used the RCS (Real-time Control System) methodology for determine these requirements.

RCS was developed by NIST for the control of intelligent systems, and has recently been used to control intelligent

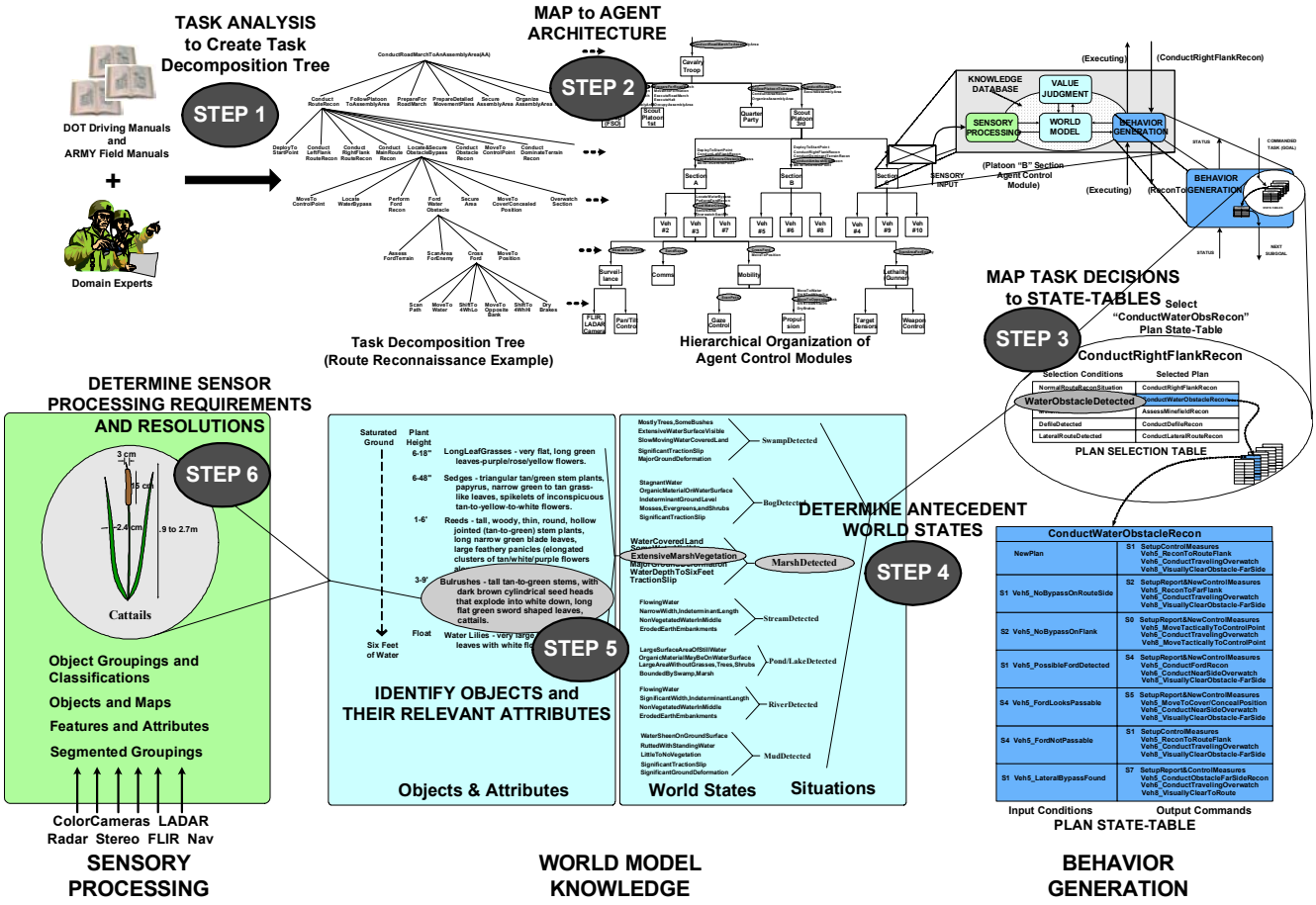


**Figure 1** - RCS Methodology

vehicles within military environment [1]. The RCS methodology concentrates on the task decomposition as the primary means of understanding the knowledge required for intelligent control. This approach is shown in Figure 1 and begins with the knowledge "mining" activities to retrieve knowledge from subject matter experts (SMEs). The gathering and formatting of this knowledge can be summarized in six steps, each of which follows from the knowledge uncovered by the previous steps:

1) The first step involves an intensive analysis of domain knowledge from manuals and subject matter experts (SMEs), especially using scenarios of particular subtask operations. The output of the effort is a structuring of this knowledge into a task decision tree form of simpler and simpler commands (actions/verbs) at simpler and simpler levels of task description.
2) This step defines the hierarchical organization of agent control modules that will execute these layers of commands in such a manner as to reasonably accomplish the tasks. This is the same as coming up with a business or military organizational structure of agent control modules (people, soldiers) to accomplish the desired tasks. This step forces a more formal structuring of all of the subtask activities as well as defining the execution structure.
3) This step clarifies the processing of each agent's input command through the use of rules to identify all of the task branching conditions with their corresponding output commands. Each of these command decompositions at each agent control module will be represented in the form of a state-table of ordered production rules (which is an implementation of an extended finite state machine (FSM)). The sequence of simpler output commands required to accomplish the input command and the named situations (branching conditions) that transition the state-table to the next output command are the primary knowledge represented in this step.
4) In this step, the above named situations that are the task branching conditions are defined in great detail in terms of their dependencies on world and task states. This step attempts to define the detailed precursor states of the world that cause a particular situation to be true.
5) In this step, we identify and name all of the objects and entities together with their particular features and attributes that are relevant to defining the above world states and situations.

6) The last step is to use the context of the particular task activities to establish the distances and, therefore, the resolutions at which the above objects and entities must be measured and recognized by the sensory processing component. This step establishes a set of requirements and/or specifications for the sensor system at the level of each separate subtask activity.

More details about this methodology can be found in [2].

The outputs of this methodology are a set of information requirements, specifically, information about tasks (step 1), agents (step 2), plans to accomplish tasks (step 3), conditions and situations (step 4), environmental entities (step 5), and attributes of those entities (step 6). These information requirements serve as the input for the ontology described in the Section 3.

### 2.2. The Scenario

This effort uses knowledge derived from the task analysis of scenarios of a light cavalry troop's execution of a Conduct Tactical Road March to Assembly Area mission, independent if the activity is performed by man or machine. In particular, we have analyzed the part of this mission that focuses on the route reconnaissance component by the Scout Platoon. This is done through scenarios that are examined at more and more detailed levels starting at the Troop Commander Level, which will perform a number of planning activities to better identify the priority information items of the route, to define the march column organization, and to specify the formation and movement technique. The Troop Commander will then dispatch a scout platoon to conduct a route reconnaissance. The scout platoon leader will do finer level planning, organizing the platoon's sections of vehicles and assigning commands to each section leader to do reconnaissance of different areas along the route while maintaining security. Each section leader will evaluate the environment to provide detailed tactical goal paths for each of his vehicles, coordinating their movement by the use of detailed motion commands to control points along with security overwatch commands. Each vehicle, in turn, performs detailed sensory processing to carry out careful analysis of the terrain in context of the mission, security, stealth, and traversability. Each vehicle then decides its optimal real-time path. If some aspect, such as a water obstacle, constrains the vehicle from following the general goal path laid out by the section leader, the vehicle does reconnaissance, moves to a secure point, and reports to the section leader. If the constraint affects the operation of the entire section (e.g. the water obstacle stretches across the entire area that the section is assigned), then the section leader coordinates his vehicles to do reconnaissance and to take up secure positions. The section leader then reports to the platoon leader.

These scenarios provide a rich set of knowledge of organizational structure, activities, commands, rules, status, sensory processing, objects and world states to be recognized, adaptation to events, and procedures required for successful execution.

### 2.3. Extracting Information Requirements

Although the RCS approach uses state tables to represent the information, the representation within the ontology does not necessarily need to be captured within state tables as long as no information is lost. Table 1 shows a small piece of a state table representation that was developed from the RCS methodology for the scenario described above. The left column shows the condition that must be true for an action to occur. The notation S# notates a state value. For example, in the first line, the system must be in state 4 and the condition "Scout Platoon Ready to Conduct Route Recon" must be true for the action "Scout Platoon Conduct Route Reconnaissance" to occur. When the action is executing, the system will change over to state 5 (S5).

**Table 1** - Excerpt from a RCS State Table

| **Conduct Tactical Road March To Assembly Area** | |
|---|---|
| **Condition** | **Action** |
| … | … |
| S4 Scout Platoon Ready to Conduct Route Recon | S5 Scout Platoon Conduct Route Reconnaissance |
| S5 Quartering Party Ready to Organize Assembly Area | S6 Quartering Party Follow Recon Platoon to Assembly Area |
| S6 Quartering Party Clear of Start Point | S7 Main Body and Trail Party Prepare for Road March |
| S7 Main Body and Trail Party Recon to Start Point Done | S8 Troop Prepare Detailed Movement Plans |
| S8 Scout Platoon Route Recon Done | S9 Scout Platoon Establish Assembly Area Security |
| S9 Quartering Party at Release Point | S10 Quartering Party Conduct Area Recon of Assembly Area |
| S10 Quartering Party Area Recon of Assembly Area Done | S11 Quartering Party Organize Assembly Area |
| S11 Quartering Party Status Assembly Area Suitable | S12 First Main Body Unit Move Into Road March Formation |
| S12 Main Body Unit At Start Point | S12 Main Body Unit Execute Tactical Road March<br>Next Main Body Unit Move Into Road March Formation |
| S12 Last Main Body Unit at Start Point | S13 Main Body Unit Execute Tactical Road March<br>Trailing Party Move Into Road March Formation |
| … | … |

## 3. THE IGV ONTOLOGY

### 3.1. Ontology Language

We decided to use the OWL-S upper ontology [8] as the underlying representation for the IGV Ontology in order, among other reasons, to document RCS in a more open XML (eXtensible Markup Language) format. OWL-S is a

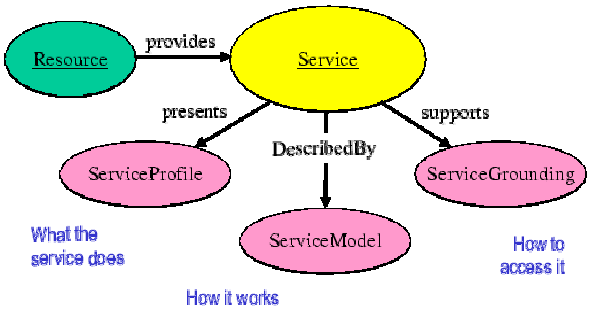


**Figure 2** - OWL-S Ontology Structure

service ontology, which supplies a core set of markup language constructs for describing the properties and capabilities of services in an unambiguous, computer-intepretable format. OWL-S, which is being developed by the Semantic Web Services arm of the DARPA Agent Markup Language (DAML) program, is based on the OWL [5]. OWL is an extension to XML and RDF (Resource Description Framework) schema that defines terms commonly used in creating a model of an object or process. OWL is a World Wide Wide Consortium (W3C) recommendation, which is analogous to an international standard in other standards bodies.

OWL-S is structured to provide three types of knowledge about a service (Figure 2), each characterized by the question it answers:

- What does the service require of the user(s), or other agents, and provide for them? The answer to this question is given in the ``profile." Thus, the class SERVICE presents a SERVICEPROFILE
- How does it work? The answer to this question is given in the ``model." Thus, the class SERVICE is describedBy a SERVICEMODEL
- How is it used? The answer to this question is given in the ``grounding." Thus, the class SERVICE supports a SERVICEGROUNDING.

Later in this paper, we will show how we use these OWL-S concepts to model a tactical behavior for an intelligent ground vehicle.

### 3.2. Tools

Before the ontology can be built, a decision was made as to which tool (or set of tools) should be used to enter, capture, and visualize the ontology. For this work, we decided to use Protégé [7]. Protégé is an ontology editor, a knowledge-base editor, as well as an open-source, Java tool that provides an extensible architecture for the creation of customized knowledge-based applications. Protégé was chosen due to its strong user community, its ability to support the OWL language (discussed below), its ease of use (as determined by previous experience), and its ability to be extended with plug-ins such as visualization tools (also discussed below).

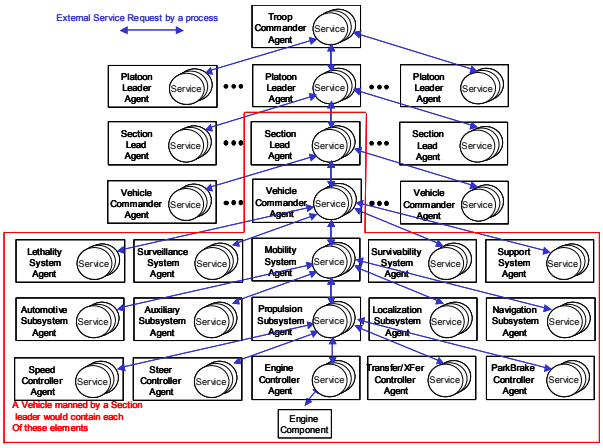


**Figure 3** - Agent Hierarchy

### 3.3. Using OWL-S To Model the Scenario

Both the RCS methodology and the OWL-S upper ontology are based on the concept of agents, service that the agents can perform, and procedures that the agents follow to perform the services. As such, there is a very clean mapping between the information that comes out of the RCS methodology and the OWL-S upper ontology. In this section, we will describe that mapping.

The first step involved setting up the agent hierarchy. In the domain we are dealing with (a light Cavalry troop), we designed an agent hierarchy as shown in Figure 3. A detailed description of this hierarchy is outside the scope of this paper. In OWL-S, we modeled all of these agents as subclasses of the IGVAgent class, which is a subclass of the ServiceResource class defined in the OWL-S upper ontology. We also specified in the constraints for each class who each agent can send external service requests to and who they can received them from.

The next step involved setting up the services and processes. Any activity that can be called by another agent is considered a service in OWL-S. Any activity that the agent performs internally that cannot be externally called is called a process. As such, we model “Conduct A Tactical Road March to an Assembly Area” as a service that is provided by a Troop agent (and can be called by a Squadron agent). The Troop agent can call services provided by other agents, such as shown by the first action in Table 1 (Scout Platoon Conduct Route Reconnaissance). In this example, we defined a service called “Conduct Route Reconnaissance” and associated that service with the Scout Platoon agent.

The state table in Table 1 shows an excerpt of the process that an agent (in this case, the Troop agent) should follow when executing the “Conduct a Tactical Road March to Assembly Area” service. This process is captured in OWL-S as a process model (as described in Section 3.1). The process model includes the steps that must be accomplished to carry out the service, and the ordering constraints on

those steps. Each step can be performed internally by the agent or could involve making an external service request (a service call) to another agent. The ordering of the steps in Table 1 is shown using the next possible state indications (S#). For example, the second to last action in Table 1 reads "S12 Main Body Unit Execute Tactical Road March, Next Main Body Unit Move Into Road March Formation." The S12 indicates what the next possible state could be after this action is executed. On the left side of the table, there are two states that start with S12 (Main Body Unit At Start Point and Last Main Body Unit at Start Point). One of these conditions has to be true after the "S12 Main Body Unit Execute Tactical Road March" action completes executing. If the last main body unit is at the start point, then the latter state will be true. If not, then the former state will be true. This continues until the latter state is true and the state table moves to the next state (S13). One can model this as a repeat-until sequence (the loop continues until the last main body until is at the start point). OWL-S provides a number of control constructs that allow you to model just about any type of process flow imaginable. We have found that the control constructs provided in OWL-S have been sufficient to model all of the behaviors we explored.

Not shown in Table 1 but explained in Section 2, conditions are a primary information output of the RCS methodology (Step 4). The series of conditions shown in Figure 1 explain how a given state in the state table is determined to be true. OWL-S 1.0 provides a stub for conditions, but does not elaborate on how they are constructed. As such, we have developed our own mechanism for representing conditions based, in part, upon Xquery and Xpath [3]. A description of the constructs used to capture the conditions is outside of the scope of this paper. OWL-S 1.1 (released in October, 2004) allows mechanisms to use external languages (e.g., Semantic Web Rules Languages (SWRL) [6], and others) to specify conditions. Future work will migrate the current condition approach to these external languages specification recommended in OWL-S 1.1.

Also not shown in Table 1 but explained in Section 2, environmental entities and their attributes are another primary output of the RCS methodology. These include other vehicles, bridges, vegetation, roads, water bodies; anything that is important to perceive in the environment relative to task that is being performed. We have started to build an environment ontology in OWL-S from the bottom up (i.e., including only entities that prove to be important based on the output of the RCS methodology). However we are currently starting to explore existing environment ontologies that currently exist to see what we can leverage.

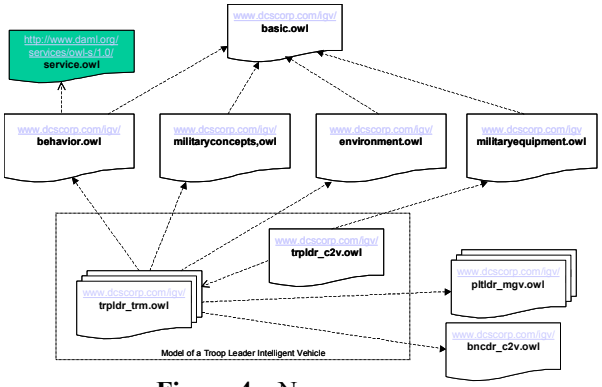


**Figure 4** – Namespaces

### 3.4. Organizing the Knowledge

Due to the sheer size of the ontology, we have taken the approach of organizing the knowledge into namespaces. A namespace is a tag prefixed to the name of the class or instance that separates the knowledge in the ontology into "pieces," where each piece represents a group of like concepts. Numerous namespaces can be imported into a single ontology and a single namespace can be reused in multiple ontologies. The contents of namespaces are often stored in a separate files.

For this effort we have identified a set of five high-level namespaces that build off of the concepts presented in OWL-S (shown in Figure 4), namely:

- Basic – data structures to capture abstract, highly reusable concepts (e.g., location, spatial relations)
- Behavior – data structures which help to describe services, agents, and conditions (e.g., and-conditions, or-conditions, external service requests)
- Military Concepts – data structures to capture common concepts within military procedures (e.g., assembly area, control points, troops)
- Environment – data structures to capture environmental concepts (e.g, water bodies, shrubs, weather conditions)
- Military Equipment – data structures to capture information about the equipment that the military uses (e.g., communication devices, weapons, measuring devices)

In addition, we have adopted the approach that we would define a separate namespace for every agents' services and processes in every tactical behavior. For example, in the "Conduct a Tactical Road March to an Assembly Area"

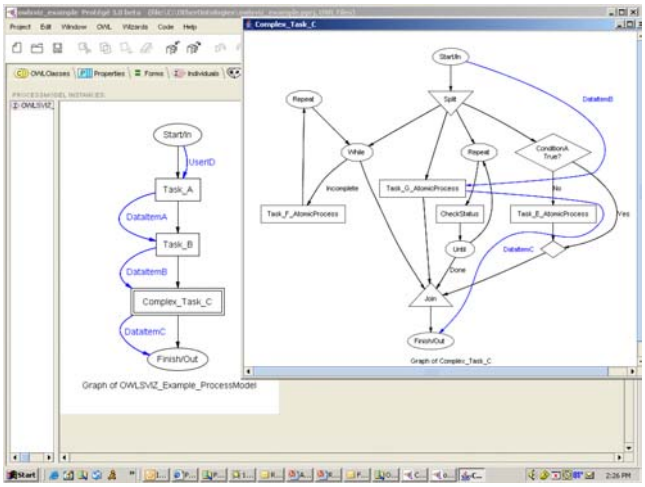

**Figure 5** - OWL-S Visualization Tool

tactical behavior, we have defined the services and corresponding process models for most of the agents shown in Figure 3. As such, we would have a namespace call Troop-ConductTacticalRoadMarchToAA, Platoon-ConductTacticalRoadMarchToAA, etc.

### 3.5. Visualization of the Ontology

While presenting this work to our funding sponsors, it quickly became obvious that we needed a better way to display the information in the ontology. A whole bunch of windows with a small amount of text in each didn't work well. As mentioned earlier, one of the reasons we chose Protégé was its ability to be extended using "plug-ins." A plug-in is a piece of code that performs a given functionality that can be incorporated into Protégé. We developed a visualization tool based upon the open source GraphViz program [4] that allows us to graph OWL-S models. A snapshot of the visualization tool, shown in Figure 5, shows process flow (dark arrows) and data flow (light arrows).

## 4. CURRENT STATUS

To date, service models have been developed for the following agents: troop, platoon, section, vehicles, mobility, propulsion, engine controller, engine, surveillance, and sensor subsystem (as shown in Figure 3). All of the service models are currently focused solely on the scenario described in Section 2.2. We have only done one strand of this scenario (e.g., we elaborated one service in each level of the hierarchy, although there were almost always many services that each agent may have to perform to accomplish the overall goal of "Conducting a Tactical Road March to An Assembly Area.") Also, this one scenario represents one of hundreds, if not thousands, of tactical behaviors that an army soldier is expected to be able to perform. It is clear that, as the details of this scenario is further modeled and more scenarios are explored, the size of the ontology will grow to be very large. At the time when this document was written, we have modeled 489 classes, 213 properties (attributes), and 2674 instances.

## 5. CONCLUSION

In this paper, we describe an ongoing effort to develop an IGV Ontology for the purpose of capturing knowledge about tactical behaviors to facilitate intelligent ground vehicle development and execution. The ontology is being built using the Protégé environment and is based upon the OWL-S upper ontology.

Through the development of this ontology, it has become apparent that the applications of a neutral and well-defined representation and tactical behaviors is far-reaching above and beyond that of controlling autonomous vehicles. Some of the potential communities that have been identified include:

- Material Acquisition Community: Documenting behavioral requirements for manned, aided, and unmanned systems; and analyzing the completeness/consistency of requirements and predicting the performance of systems developed to meet those requirements.
- Vendor Community: Unambiguously interpreting behavioral requirements; and rapid development of simulation/prototypes to evaluate design alternatives.
- Training Community: Availability of machine-readable sources of tactical knowledge for automatic generation of training materials and training scenarios

Initial results of this effort have shown that:

- Ontologies appear to be an excellent approach for capture information in a computer-interpretable format about tactical behaviors
- The RCS methodology has provided information requirements to the ontology at a level of detail that greatly facilitates the ontology development process. A large part of building any type of ontology is extracting the information requirements that must be represented. The RCS methodology directly addresses this need.
- OWL-S provide a very nice mapping to the RCS methodology and serves as an excellent upper ontology to represent tactical behaviors

## ACKNOWLEDGMENT

This work was sponsored by U.S. Army Tank Automotive Research, Development and Engineering Center (TARDEC).

## REFERENCES

1. Albus, J., "4-D/RCS: A Reference Model Architecture for Demo III," *NISTIR 5994*, Gaithersburg, MD, 1997.
2. Barbera, T., et. al., "Software Engineering for Intelligent Control Systems," *Kunstliche Intelligenz Journal: Special Issue on Software Engineering for Knowledge-based Systems*, Vol. 3, No. 4, 2004.
3. Boag, S., et. al., "XQuery 1.0: An XML Query Language," *W3C Web page: http://www.w3.org/TR/xquery/*, 2004.
4. Gansner, E. and North, S., "An Open Graph Visualization System and Its Applications," *Software - Practice and Experience*, Vol. 00, No. S1, 1999.
5. Harmelen, F. and McGuiness, D., "OWL Web Ontology Language Overview," *W3C web site: http://www.w3.org/TR/2004/REC-owl-features-20040210/*, 2004.
6. Horrocks, et. al.., "SWRL: A Semantic Web Rule Language Combining OWL and RuleML: Version 0.5," *http://www.daml.org/2003/11/swrl/*, 2003.
7. Schlenoff, C., Washington, R., and Barbera, T., "Experiences in Developing an Intelligent Ground Vehicle (IGV) Ontology in Protégé," *The 7th Int. Protégé Conference*, Bethesda, MD, 2004.
8. The OWL Services Coalition, "OWL-S 1.0 Release," http://daml.org/services/owl-s/1.0/owl-s.pdf, 2003.