An Investigation of Machine Learning Techniques for Use in Training Agents for Military Simulations

MICHAEL R. HIEB AND J. MARK PULLEN

Abstract – **Agents assist users with performing tasks in computer-based applications. The current practice of building an agent involves a developer programming it for each task it must perform, but agents constructed in this manner are difficult to modify and cannot be trained by a user. Agent-Disciple is a system for training *instructable* agents through user-agent interaction. In Agent-Disciple a user trains an instructable agent through the interface of the user's application by providing specific examples of tasks and their solutions, explanations of these solutions and supervises the agent as it performs new tasks.**

We report here on our work that uses Agent-Disciple to provide a learning agent that can command simulated military forces. Military simulations currently have many limitations in modeling human behavior. While it is relatively straightforward to build models of doctrine, it is difficult to have agents utilize this doctrine in varying contexts. There are too many factors to consider when building deterministic models of behavior, even in well-defined situations. We applied Agent-Disciple to circumvent this problem by using heuristic learning methods.

A case study is presented in developing an instructable Company Commander Agent for the Modular Semi-Automated Forces (ModSAF) simulation – a state-of-the-art, real-time, distributed interactive military simulation currently utilized in large-scale training exercises. A ModSAF user can train the Company Commander Agent interactively, using the ModSAF interface, to perform various military missions using the Captain system based on Agent-Disciple. A training session with the agent illustrates the different types of learning interactions available in Agent-Disciple.

I. INTRODUCTION

Computer Generated Forces (CGF) are simulations of military entities. The ability to build intelligent command agents for CGF is significantly constrained by the knowledge acquisition effort required. Many iterations by Subject Matter Experts (SME), programmers and knowledge

This research was conducted in the Center of Excellence in Command, Control, Communications & Intelligence and the Computer Science Department at George Mason University. Work on ModSAF was sponsored in part by DMSO under DISA contract DCA100-91-C-0033 and work on Disciple was sponsored in part by the Defense Advanced Research Projects Agency under Contract No. N66001-95-D-8653.

M. R. Hieb is with AB Technologies, 1901 N. Beauregard St., Alexandria, VA, 22311.

J. M. Pullen is with the Department of Computer Science, George Mason University, Fairfax, VA, 22030.

engineers are required to develop acceptable behavior even for a narrow range of situations. Moreover, once built the agents cannot adapt themselves to changes.

Various automated knowledge acquisition tools have been proposed and utilized for this problem, but there is no standard acquisition methodology for CGF that has gained acceptance. The existing approaches primarily utilize programmers and knowledge engineers to encode the expertise of a SME. Our goal is to change this paradigm by enabling the SME to use a familiar simulation interface to instruct an agent directly under the Agent-Disciple system. We use the ModSAF package developed for the U.S. Army to execute the agent's orders. Direct instruction reduces the involvement of programmers and knowledge engineers, increasing the efficiency of the acquisition process and improving the quality of the acquired knowledge.

Agents that learn from Agent-Disciple function as instructable ModSAF agents. This approach provides a new approach to solving the knowledge acquisition problem for CGF. Agent-Disciple follows a general methodology for developing instructable agents for existing applications given in [4]. The process utilizes Programming by Demonstration (PBD) [2] and Machine Learning techniques to allow instruction by an SME. PBD systems give an end user the ability to create programs by demonstrating their actions thorough a graphical user interface. PBD is a new research area that is concerned with interactive learning of user tasks from only a few examples and explanations given by the user. By contrast, Machine Learning as employed in Agent-Disciple uses more formal, domain-independent autonomous learning methods. Often the inputs to machine learning programs are large numbers of examples, extensive background knowledge, or, for multistrategy learning systems, both.

Our system is called Captain. Fig. 1 shows its system design. An SME uses editors within ModSAF to teach a command agent new tasks. The machine learning is performed on a separate workstation running Agent-Disciple software (Agent-Disciple provides the learning functions of Disciple in a modular toolkit). The gray modules indicate modules that were modified for Captain.

In our system, an SME teaches a ModSAF agent through the ModSAF Graphical User Interface rather than using a different interface for the learning system. The SME initially demonstrates to the ModSAF agent how to perform a new mission. The SME uses the existing ModSAF task editors to "program" the agent, as the SME normally would, creating a sequence of specific tasks. This sequence is given to the learning system as an initial example of the mission. The SME then explains the relevant features of the mission. The learning system will then attempt to perform a different instance of the mission (e.g. on a different piece of terrain) under the supervision of the SME, asking the SME to classify its solution of the mission as a correct or incorrect example. The SME uses ModSAF's graphical user interface to correct the agent if it does not perform the mission as required by the SME. After this teaching session, the

Agent-Disciple agent will have learned how to perform this type of mission (i.e., create a rule specifying how to select tasks and instantiate task parameters for a specific mission) and be able to perform this new mission when asked to do so by the SME. Thus the SME *actually will perform the programming*, but by example rather than by using the task editors.

We prototyped this approach with the Captain system [4], [5], [6], consisting of an integration of the apprenticeship learning system, Disciple [16], and ModSAF. In Captain, a ModSAF company commander can be taught how to place its platoons to defend its assigned area of responsibility. This process involves eliciting an initial example from the SME, eliciting 5 to 10 explanations and showing the SME 5 to 10 examples of solutions that the system generates. Experiments with Captain indicate that this type of system will prove scalable as it is applied to learning other tasks in this domain. We present an analysis of our learning method in Section V that supports this conclusion, but note that since the system is based on user interaction, it is difficult to formally prove scalability. This is an area for future research with Captain.

Captain utilized the ModSAF terrain map to show examples to the SME for classification, but did not fully integrate other learning interactions (specifying examples and explanations) into the ModSAF interface. Thus the experiments were run with two interfaces – 1) the Captain textual interface for SME interaction; and 2) the ModSAF graphical interface for display of the example placements on the digital terrain map.

Like much other work in Machine Intelligence technology, Captain is based more on heuristics than on derived principles. However, as we describe below, military subject matter experts have reviewed the results of the work presented here and confirmed that the Captain system learned to perform on a par with the SMEs who trained it. Therefore we are offering this description of what we believe to be a highly successful example of the heuristic approach.

The remainder of this paper is organized as follows: Section II presents a discussion of related work. Section III explains representations that Agent-Disciple uses for learning. Section IV describes a case study of building a military command agent with Agent-Disciple. Section V concludes the paper with a discussion of our agent-building approach and future plans.

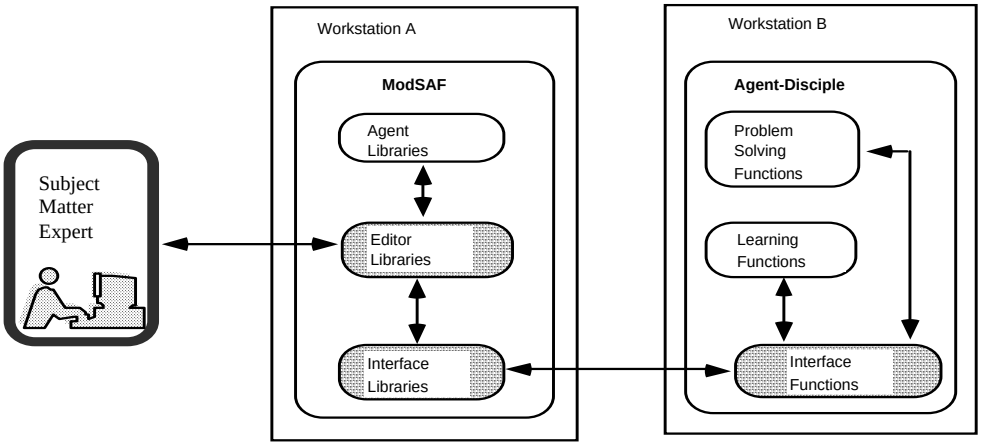

Fig. 1. Captain Design.

II. RELATED WORK

We begin with a review of some of the related research on learning, particularly Apprenticeship learning and the new field of Programming by Demonstration. Then we describe how this research applies to agents.

A. Approaches to Machine Learning

Apprenticeship Learning systems are at the intersection of the fields of Machine Learning and Knowledge Acquisition. An Apprentice Learning System can be defined as an interactive knowledge-based consultant that is provided with an initial domain theory and is able to assimilate new problem-solving knowledge by observing and analyzing the problem-solving steps of its users through their normal use of the system [20].

Apprenticeship Learning systems involve the user in the learning process, where Machine Learning systems generally are not interactive. In Apprenticeship Learning systems the user provides the learning system's input in a representation that is natural to the user. The learning system has an interaction with the user during the learning process, where the user may be asked to give other examples, confirm a hypothesis, or give explanations. The output of this learning generally is presented to the user prior to being translated into a form usable by a performance element (e.g., a rule-based production system) [18]. Most Machine Learning systems require their learning input to be put into a special format. The user may not be able to understand the input (which may be in the form of data) or the output (which is often in the form of rules) unless the user is quite familiar with the learning method.

Knowledge Acquisition systems and Apprenticeship Learning systems are closely related. However, the emphasis of most Knowledge Acquisition systems is on modeling the initial

knowledge base and eliciting knowledge. The emphasis in Apprenticeship Learning systems is on refining knowledge that has already been elicited or created, using Machine Learning techniques that learn from a user.

Programming by Demonstration (PBD) systems are a type of Apprenticeship Learning systems that give an end user the ability to create programs by demonstrating their actions. Machine Learning covers an overlapping area of research concerned with methods that learn concepts from example or domain theories, including such instruction from a teacher. An example of a PBD system is the Metamouse system [14] which gives the user the ability to automate drawing tasks. The user instructs an agent (a turtle named Basil) on how to manipulate objects through an innovative graphical interface. Basil learns from specification of graphical constraints to construct a program that automates graphical editing. The program can have loops and conditionals.

PBD systems are significant because their goal is to empower the end user by assuming that, if a user knows how to perform a task on the computer, then that knowledge should be sufficient to create a program to perform the task. Rather than learn a programming language, the user should be able to instruct the computer to watch as the task is demonstrated [2]. A common concern among PBD systems is interface design. The graphical (and verbal) user interface of these systems is generally very sophisticated, and the inferencing techniques are usually more specific to the task domain than machine learning methods [13].

PBD systems generally deal with the automation of simple tasks. They generally do not deal with automating complex tasks or behaviors, such as concerns the ModSAF agents. The systems do not provide facilities for the end user to specify domain knowledge to the system, as is done with knowledge elicitation or knowledge acquisition methods.

B. Instructable Agents

Software agents are programs that can execute with their own identity within an application, either autonomously or semi-autonomously. The agents that are being developed today either have fixed (non-adaptive) behavior or can exhibit some limited forms of learning. Agent-Disciple can be thought of as an agent development environment either for training existing agents or for building entirely new agents.

ModSAF agents use a task-level architecture similar to a subsumption architecture [1]. This allows a user to give orders to an agent, which then attempts to carry out the orders, unless it reacts to a condition for which it was programmed (e.g., a threat). Since this is a reactive architecture, the agents must be supervised closely by the SME.

Another approach to CGF development has been to develop entirely new ModSAF agents using the Soar problem-solving model [15}. These agents currently operate primarily in

aviation environments. The Soar-based agents have the potential to significantly improve the behavior of ModSAF entities, but conducting the knowledge acquisition to build such agents remains a difficult problem because they do not have the ability to learn.

Soar [11] is a general problem-solving architecture that addresses the problem of agent learning. Soar has a single learning mechanism that is integral to its architecture: chunking. By contrast, Agent-Disciple integrates many learning methods for agent instruction. Soar has a very elaborate model of problem solving – the Problem Space Computational Method (PSCM) – that uses deductive rules. Agent-Disciple uses rules with plausible conditions and is able to reason with incomplete knowledge, although its problem-solving model is also more complicated than most expert systems.

Huffman [10] used Soar in his system Instructo-Soar, where an instructable agent learns from tutorial instruction. Instructo-Soar learns general knowledge from specific instructions, using rote learning and a type of inductive learning (situated explanation) in addition to the chunking of Soar. Huffman delineates the type of knowledge that must be learned to build an agent in Soar's computational model, and demonstrates the ability of Instructo-Soar to acquire the majority of knowledge types necessary. Instructo-Soar has been demonstrated in a small domain, a blocks world with a small number of operators, properties and relationships (less than 10 of each).

III. PLAUSIBLE VERSION SPACE RULES

Here we first describe the novel structure of rules in Agent-Disciple and then give examples of how Agent-Disciple uses such plausible version space rules.

A. Plausible Version Space Rules

Agent Disciple uses a hybrid knowledge representation integrating semantic networks and rules. Semantic networks represent information from a terrain database at a conceptual level, as well as knowledge about forces and weapon systems. In order to facilitate learning, the objects and the rules both use the following representation unit:

```
(concept-i  concept-k      (FEATURE-1 value-1)
                           . . .
                           (FEATURE-n value-n))
```

This expression defines 'concept-k' as being a subclass of 'concept-i' (from which it inherits features) with additional features. The value of a feature may be a constant or another concept.

In Agent-Disciple, rules are procedures that consist of a PROBLEM statement, CONDITIONS and a SOLUTION statement. Each condition (also called a *clause*) consists of a plausible upper bound and plausible lower bound that are in the format of the representation unit described above. The plausible upper bound is a conjunctive expression that is supposed to be more general than the exact condition, and the plausible lower bound is a conjunctive expression that is supposed to be less general than the exact condition. The two bounds define a *plausible version space* (PVS) for the condition to be learned by Disciple [17]. The bounds and the version space are called plausible because the learning process takes place in an incomplete representation language that may cause them to be inconsistent (a lower bound that covers some negative examples or an upper bound that does not cover all positive examples).

Fig. 2 shows the general form of a PVS procedure in Agent-Disciple. A procedure is learned from specific problem solving episodes indicated by a user. Once learned, a procedure can be selected to be performed by an SME. A *mission* is a goal specification given to the agent (the agent in the military simulation is given an order). A *task* is an action that the agent can take in the simulation.

The terms $\mathbf{p_1}$ through $\mathbf{p_n}$ and $\mathbf{p_{11}}$ through $\mathbf{p_{iv}}$ represent parameter names, $\mathbf{m_n}$ through $\mathbf{m_n}$ represent mission parameters, $\mathbf{t_{11}}$ through $\mathbf{t_{iv}}$ represent task parameters, um and ut represent the upper bound class, lm and lt represent the lower bound class, and cm through ct represent constraints upon the parameters. The plausible upper bound and plausible lower bound are both conjunctive expressions. The plausible upper bound is more general than the plausible lower bound. The upper bound represents a set of possible solutions, while the lower bound represents the least general generalization of the set of solutions actually encountered.

PROBLEM:
to accomplish
MISSION $\mathbf{p_1}$ $\mathbf{m_1}$ ... $\mathbf{p_n}$ $\mathbf{m_n}$

CONDITIONS:

	plausible upper bound	plausible lower bound
verify		
	(um_1 $\mathbf{m_1}$ cm_{11} ... cm_{1o}) &	(lm_1 $\mathbf{m_1}$ cm_{11} ... cm_{1o})&
	...	...
	(um_n $\mathbf{m_n}$ cm_{n1} ... cm_{np})	(lm_n $\mathbf{m_n}$ cm_{n1} ... cm_{np})
find		
	(ut_{11} $\mathbf{t_{11}}$ ct_{111} ... ct_{11r}) &	(lt_1 $\mathbf{t_{11}}$ ct_{111} ... ct_{11r}) &
	...	...
	(ut_{iv} $\mathbf{t_{iv}}$ ct_{iv1} ... ct_{ivs})	(lt_1 $\mathbf{t_{iv}}$ ct_{iv1} ... ct_{ivs})

SOLUTION:
perform
$TASK_1$ $\mathbf{p_{11}}$ $\mathbf{t_{11}}$ ... $\mathbf{p_{1w}}$ $\mathbf{t_{1u}}$
...
$TASK_i$ $\mathbf{p_{i1}}$ $\mathbf{t_{i1}}$ ... $\mathbf{p_{iv}}$ $\mathbf{t_{iv}}$

Fig. 2. PVS Procedure.

B. Examples of Using Plausible Version Space Rules

Fig. 3 contains simple procedures for an agent corresponding to the situation depicted in Fig. 4. Procedure P1 specifies how to OBSERVE an object **t1**: verify that it meets the constraint in the lower bound – that it is a terrain-element (both the upper and lower bounds are the same in this case); if it is, then find objects for the task parameters **t2** and **u** subject to the constraints in the lower bound – that **t2** is a hill opposite from **t1** and that **u** is an armored-platoon; if there are no armored platoons, then use the upper bound and attempt to find a platoon; if objects for the task parameters are found, then perform the march task.

Procedure P1 has been learned from the following initial example[1].

to accomplish
OBSERVE
  TERRAIN hill-60-70
perform
MARCH
  UNIT-ID platoon-a1 LOCATION hill-44-91

The initial example is expressed as a tuple,

(**t1** ← hill-60-70, **t2** ← hill-44-91, **u** ← platoon-a2)

[1] An instructor gives the initial example and classifies the subsequent examples in this scenario as positive or negative. This scenario focuses on the learning method rather than interaction.

A detailed description of how the upper and lower bounds are formulated and modified to obtain P1 is given in [4]. The second example,

(**t1** ← lake-57-82, **t2** ← hill-60-70, **u** ← platoon-a2)

is positive and indicates that the lake in Fig. 4 can be observed from hill-60-70 by a platoon. The example

(**t1** ← hill-44-91, **t2** ← lake-57-82, **u** ← platoon-a2)

is negative since the platoon cannot move onto the lake (the platoon is a motorized unit with tracked vehicles and cannot drive on the lake). The example

(**t1** ← hill-60-70, **t2** ← hill-44-91, **u** ← company-a)

is negative since a company cannot be utilized as the observing unit (it is too large to perform the observation mission).

After the procedure is learned, the agent can use it as follows:

1. SELECT – The agent selects a procedure to accomplish a specific mission and binds the variables in the problem to the mission parameters.
2. VERIFY – Verify that the mission parameters meet the constraints imposed by the corresponding *verify* lower bound conditions.
3. FIND – Find a set of objects corresponding to the task parameters that meet the constraints in the *find* lower bound conditions.
4. EXECUTE – Instantiate the task(s) in the solution with the set of objects from steps 2 & 3 corresponding to the parameters of the task(s), and invoke the task(s).

P1:

to accomplish
OBSERVE TERRAIN **t1**

plausible lower bound	plausible upper bound
verify	
(terrain-element **t1**)	(terrain-element **t1**)
find	
(hill **t2** (OPPOSITE **t1**))	(hill **t2** (OPPOSITE **t1**))
(armored-platoon **u**)	(platoon **u**)

perform
MARCH UNIT-ID **u** LOCATION **t2**

with the positive examples
(**t1** ← hill-60-70, **t2** ← hill-44-91, **u** ← platoon-a2)
(**t1** ← lake-57-82, **t2** ← hill-60-70, **u** ← platoon-a2)

with the negative examples
(**t1** ← hill-44-91, **t2** ← lake-57-82, **u** ← platoon-a2)
(**t1** ← hill-60-70, **t2** ← hill-44-91, **u** ← company-a)

P2:

to accomplish
MOVE UNIT-ID **c** LOCATION **t**

plausible lower bound	plausible upper bound
verify	
(company **c** (COMMANDS **p1**) (COMMANDS **p2**) (COMMANDS **p3**))	(company **c** (COMMANDS **p1**) (COMMANDS **p1**) (COMMANDS **p3**))
(hill **t**)	(terrain-element **t**)
find	
(armored-platoon **p1**)	(platoon **p1**)
(armored-platoon **p2**)	(platoon **p2**)
(infantry-platoon **p3**)	(platoon **p3**)

perform
MARCH UNIT-ID **p1** LOCATION **t**
MARCH UNIT-ID **p2** LOCATION **t**
MARCH UNIT-ID **p3** LOCATION **t**

with the positive examples
(**t** ← hill-60-70, **c** ← company-a, **p1** ← platoon-a1, **p2** ← platoon-a2, **p3** ← platoon-a3)
(**t** ← hill-44-91, **c** ← company-h, **p1** ← platoon-h7, **p2** ← platoon-h8, **p3** ← platoon-h4)

Fig. 3. Example of Plausible Version Space Procedures

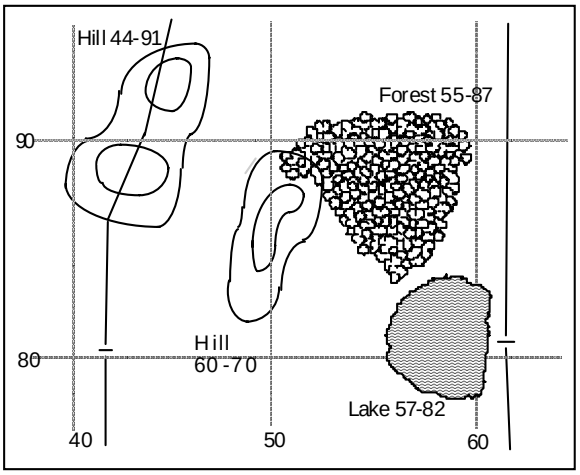

Fig. 4 Terrain Map

If the agent cannot find a procedure in step 1 to accomplish the mission, or the mission parameters do not meet the constraints imposed by the verify lower bound conditions in step 2, or the agent is unable to find a solution in step 3 (a set of objects meeting the constraints in the find lower bound conditions), then the agent will be unable to accomplish the mission. To simplify problem solving, only the lower bound is used. The upper bound is manipulated during learning and is kept so that the rule can be modified later.

For example, the user of the simulation may wish to have the agent monitor for enemy activity in the area depicted by Fig. 4. The user selects the agent and orders it to OBSERVE forest-55-87.

1. SELECT – The agent selects P1 to OBSERVE forest-55-87 and binds the variable **t1** to the object forest-55-87.
2. VERIFY – The agent checks that the object represented by parameter **t1** meets the constraints imposed by the *verify* lower bound condition in P1:
   (terrain-element **t1**)
   Since forest-55-87 is a terrain-element, **t1** is verified.
3. FIND – Find the objects corresponding to **t2** and **u** that meet the constraints in the *find* lower bound in P1:
   (hill **t2** (OPPOSITE **t1**))
   (armored-platoon **u**)

   t1 was bound to the object forest-55-87 in step one. The object found for **t2** must be a hill opposite from the object represented by **t1**. This must be hill-60-70 according to the knowledge in the agent's semantic network. Then the object found for **u** must be an armored platoon. platoon-a2 is found, but any other armored platoon would satisfy the constraint.
4. EXECUTE – The agent orders platoon-a2 to execute the march task to hill-60-70.

Similarly, procedure P2 has been learned from the following initial example:

to accomplish
MOVE
  UNIT-ID company-a LOCATION hill-60-70
perform
MARCH
  UNIT-ID platoon-a1 LOCATION hill-60-70
MARCH
  UNIT-ID platoon-a2 LOCATION hill-60-70
MARCH
  UNIT-ID platoon-a3 LOCATION hill-60-70

This procedure specifies how to move a company to a position – by moving each of the platoons associated with that company using the appropriate movement task for a platoon (MARCH). The relationship COMMANDS must hold since otherwise a company could "take" another company's platoons. There are two positive examples, and the procedure is less completely learned than procedure P1. For example, the variables representing the platoons in the lower bound could be further generalized from armored-platoon & infantry-platoon to platoon, since it does not matter what type the platoons are.

When performing procedure P2, there is an additional complication since the constraints on mission parameter **c** involve other variables representing task parameters from the solution. In this case a least commitment strategy is used in the VERIFY step, where it is verified that the object represented by **c** has the relationship COMMANDS to three other objects, without specifying what these objects are. These objects are then found in the FIND step.

IV. CASE STUDY USING CAPTAIN

The design of Captain follows the Agent-Disciple methodology for creating an Agent Training Environment given in [4]. The basic learning and problem-solving functions were taken from the Agent-Disciple toolkit. The overall implementation is depicted in Fig. 5.

A. Distributed Interface to Learning System

Researchers in the PBD field have found that it is very difficult to interface learning systems to existing applications. We have developed a distributed interface that allows the very different Disciple and ModSAF to work together. This approach has the advantage of allowing these two computationally intensive systems to run on separate processors.

Table 1 shows the interface protocol created for the interface. The phases correspond to distinct sets of learning functions as in [4]. Within the phases, types are discrete events, triggering actions. Even types indicate ModSAF sending data to Disciple, while odd types indicate Disciple sending data to ModSAF.

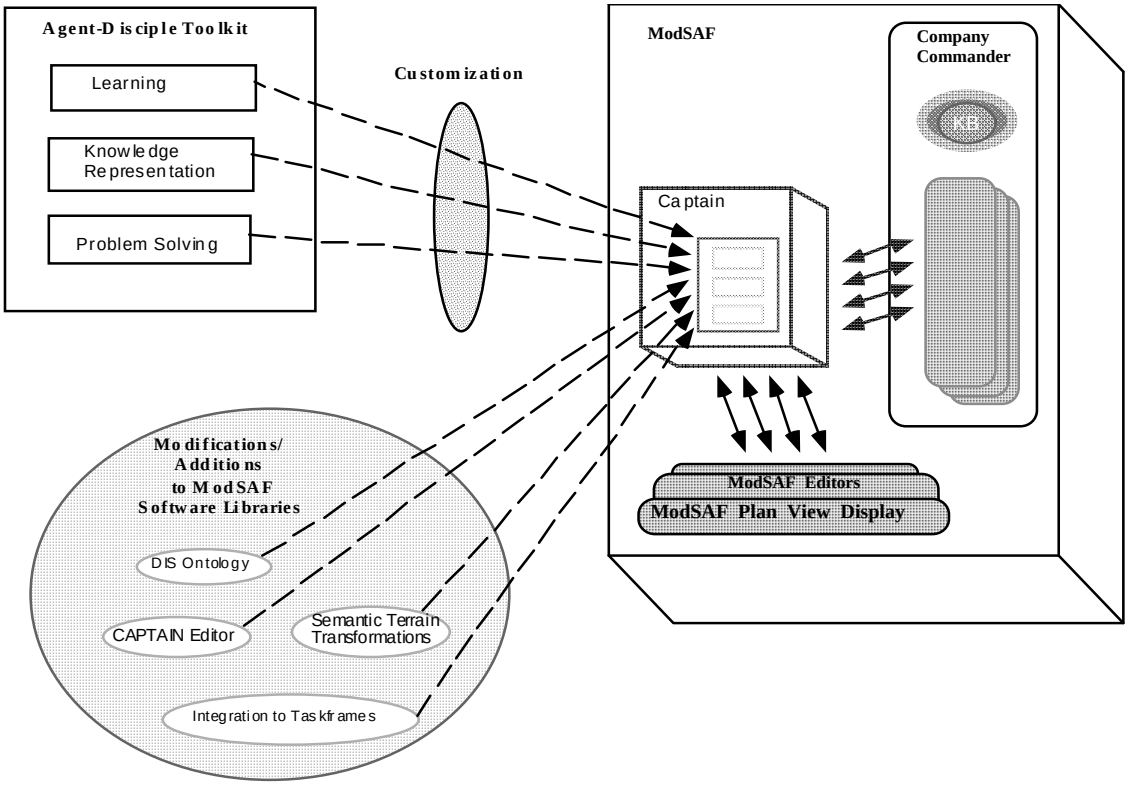

Fig. 5:. Constructing Captain Using the Agent-Disciple Toolkit.

B. Training A ModSAF Agent

This section illustrates the training of an instructable agent once the Agent Training Environment has been constructed. In the training scenario presented, an instructor uses Captain to teach a ModSAF agent a new mission. After the agent learns the new mission by formulating a procedure, it applies it to a new situation. An analysis of the learning interaction and efficiency follows. (See Appendix 1 for a glossary of military terms.)

Scenario

A ModSAF company commander is given an order to defend an area. A company commander in ModSAF commands 3 to 4 platoons. Each armored platoon is composed of 3 to 4 tanks. Each infantry platoon is composed of 3 dismounted infantry squads of 12 soldiers. The company commander is given an *area of responsibility* and an *avenue of approach*. The semantic terrain transformations will determine the optimum *engagement area* on the avenue of approach upon which the units will coordinate their fire. These three features are drawn on an overlay to

Specification of Initial Scenario Signals

Phase 0 - Give Initial Example
- Type 0 - Select Example Template
- Type 1 - Send Acknowledgement of Template
- Type 2 - Select Initial Example
- Type 3 - Send Acknowledgement of Initial Example

Phase 1 - Give Initial Explanations
- Type 0 - Select Variable to Generate Explanations from
- Type 1 - Send List of Explanations
- Type 2 - Select Explanation(s)
- Type 3 - Send Acknowledgement of Explanations

Phase 2 - Quit Initial Signal Phases
- Type 0 - Quit Specification of initial
- Type 1 - Send Acknowledgement of Quit

Learning through Experimentation Signals

Phase 3 - Experimentation Search Parameters
- Type 0 - Select Variable(s) to Fix
- Type 1 - Send Acknowledgement Fixed
- Type 101 - Quit Search Phase
- Type 102 - Send Acknowledgement of Quit

Phase 4 - Generate Experimentation Example
- Type 0 - Request Example to be Generated
- Type 1 - Send Example

Phase 5 - Give Experimentation Example
- Type 0 - Specify Example
- Type 0 - Send Acknowledgement of Example

Phase 6 - Classify Experimentation Example
- Type 0 - Classify Example
- Type 1 - Send Acknowledgement of Classification

Phase 7 - Explain Mistake in Experimentation
- Type 0 - Select Variable to Generate Explanations from
- Type 1 - Send List of Explanations
- Type 2 - Select Explanation(s)
- Type 3 - Send Acknowledgement of Explanations
- Type 4 - Select Variable to Blame
- Type 5 - Send Acknowledgement
- Type 101 - Quit Explanation Phase
- Type 102 - Send Acknowledgement

Phase 8 - Quit Experimentation Signal Phases
- Type 0 - Quit Experimentation
- Type 1 - Send Acknowledgement of Quit

Table 1. Protocol for Agent-Disciple / ModSAF Interface.

the terrain in Fig. 6. The company defensive mission requires determination of the best available terrain positions for cover and concealment, coordination of fire, ability to retreat, and many other factors. Along with terrain selection, units must be selected to place upon the appropriate terrain, since different platoons have different capabilities. The solution to the mission is a position for each element of the platoon is called a *placement*.

When Captain was implemented, there was no ModSAF task directly corresponding to a defend-area mission at any echelon level. The closest task is "Hasty Occupy Position", but that task requires a human commander, at a minimum, to place each platoon manually and indicate a point where the platoons will coordinate their fire when the enemy approaches. The commander can specify other parameters for better performance.

If an order is given for this class of mission, and the placement is determined, the platoons of the company are moved into their defensive positions. If the placement is good, then the approaching enemy will not be able to force a way through the area. A good placement usually results in the enemy not sensing the defenders until they have entered the area of engagement.

Fig. 6 shows a company, Company D, defending a valley with 2 armored platoons and one infantry platoon[2]. Two enemy companies are approaching the valley along the expected avenue of approach, the road. In this actual ModSAF simulation scenario, all of the enemy forces are stopped in the valley, at a cost of approximately 25% losses in Company D. ModSAF is a real time, non-deterministic simulation, and there is variability in the results. However, if

[2] The infantry platoons are slightly to the northwest of hill-sector 868-1 and are shown as a clump of very small circles.

the platoons of Company D are not well-placed, then they are invariably destroyed and do not stop the enemy.

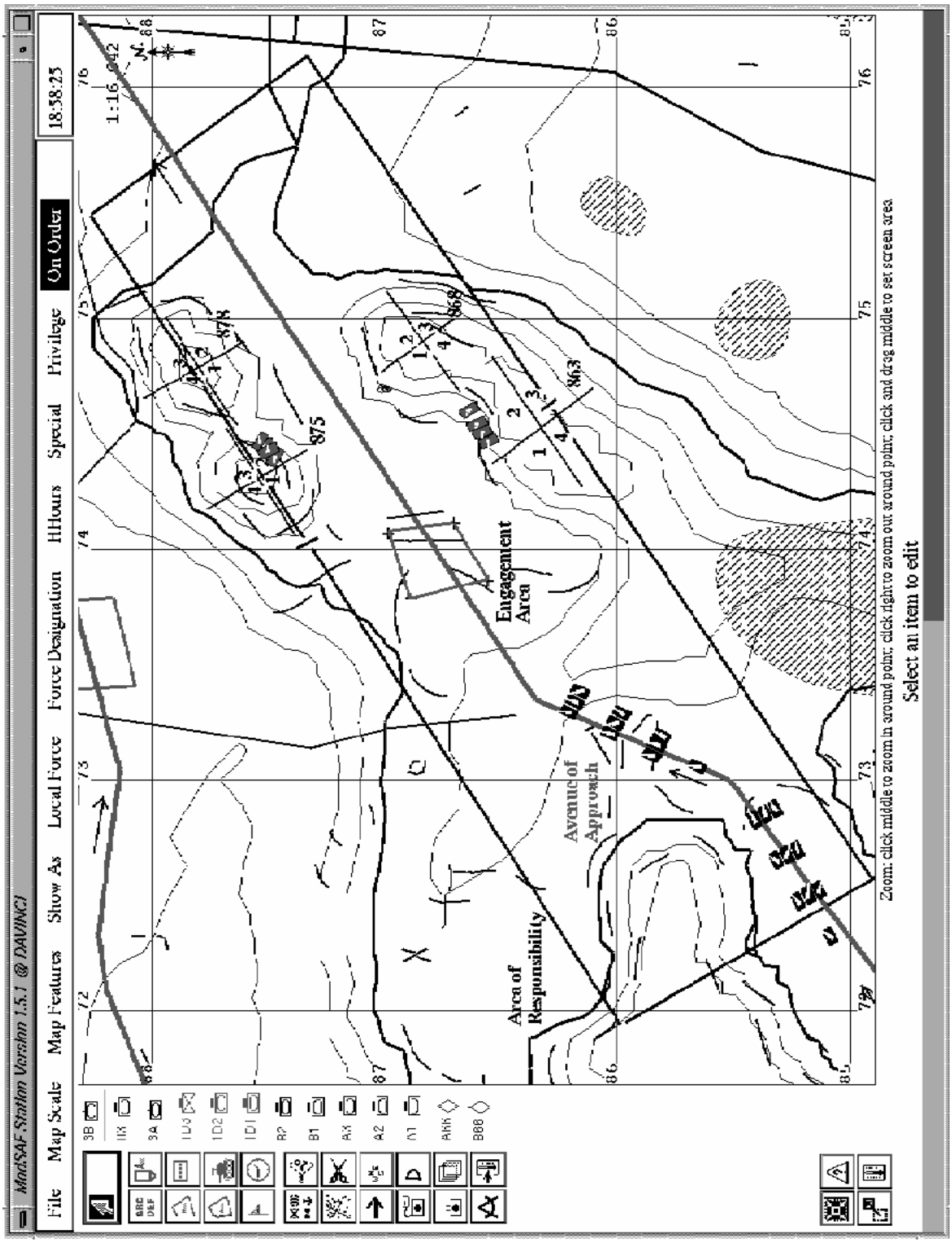

Fig. 6. ModSAF Defensive Mission Scenario.

Initial Example

The user initiates the instruction session by showing the ModSAF company agent a specific example of a correct placement. The user places three platoons of Company D (D1, D2 & D3) on the ModSAF map to defend the company's area of responsibility, as indicated in Fig. 7. The user uses the ModSAF simulation interface as the user normally would when orienting units. Fig. 8 shows the textual representation of the example mission (the problem) and also the solution. The objects already have been turned into variables as indicated by the variables paired with the objects. The variables will be utilized in the next section to generate explanations. The actual formatting of the example was edited to improve its readability. The structure of the problem and solution was preset to the defensive mission task. The system maintains a correspondence between each concept in the textual representation (e.g. hill-sector-868-1) and the corresponding object (region) on the map.

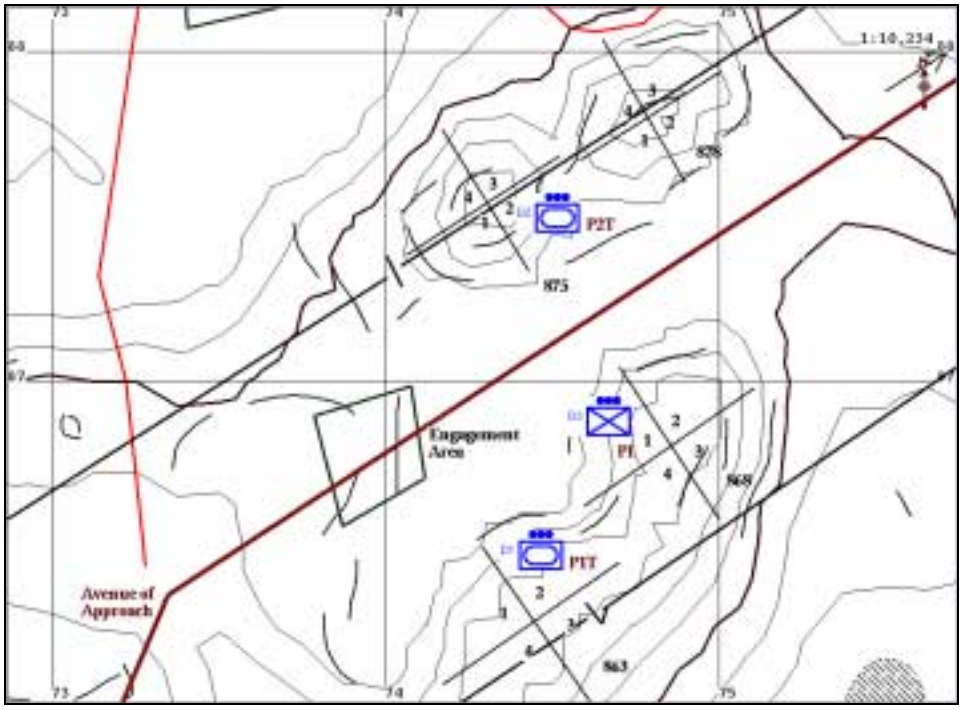

Fig. 7. Initial Placement on ModSAF Map by User.

Initial Explanations

Along with the initial example, the user is asked for explanations of why the indicated solution is correct. The user can generate plausible explanations using the Captain interface. Captain uses several heuristics to propose partial plausible explanations from which the user is requested to choose the relevant ones, as shown in Appendix B. The partial explanations proposed by the system are relationships between the objects from the problem and its

solution, or properties of these objects that are represented in the agent's knowledge base. For instance, in the case of the example considered, they are relationships between the platoons and the terrain features.

to accomplish:

POSITION-COMPANY	UNIT-ID (company-D **c**)
	MISSION (defend-area-mission-D **m)**
	LOCATION (company-D-area-of-responsibility **ar)**
	ENEMY-ORIENTATION (avenue-of-approach-D **av)**
	ENGAGEMENT-AREA (engagement-area-D **e**)

perform:

POSITION-PLATOON	UNIT-ID (platoon-D3 **Pl**)
	LOCATION (hill-sector-868-1 **hsi**)
POSITION-PLATOON	UNIT-ID (platoon-D1 **P1T**)
	LOCATION (hill-sector-863-2 **hs1tp**)
POSITION-PLATOON	UNIT-ID (platoon-D2 **P2T**)
	LOCATION (hill-sector-875-2 **hs2tp**)

Fig. 8. Initial Placement Problem / Solution.

There are several general explanation patterns in Captain. These are matched against the knowledge base to generate specific plausible explanations. The user guides the process of generating explanations by selecting an object from the problem or solution, indicated by its associated variable. As indicated below, the user specified 5 variables to generate 25 explanations from which the user chose 14 as relevant. The chosen explanations indicate that it is important that this is a defensive area mission for Company D, that the platoons to be placed belong to Company D, and that these platoons are placed in Company D's area of responsibility, i.e. in positions where they can see the engagement area.

While it is important to have some explanations of the initial example, there is no requirement that a complete set of explanations must be specified. Indeed, the assumption made by Captain is that this initial explanation set is incomplete (and possibly even incorrect) and will be completed during experimentation. Consequently, the initial example from Fig. 7 will be available for the entire duration of the learning session so that the agent can ask additional questions about this example.

The relevant explanations identified by the user are used by the agent to generate an initial plausible version space for the procedure to be learned. This version space is indicated in Fig. 9, but is not shown to the user who communicates with the system only through concrete examples and explanations. The conclusion of the procedure in Fig. 9 is obtained in Captain by turning the objects from the initial example (Fig. 8) into variables.

To Accomplish
POSITION-COMPANY UNIT-ID **c**
MISSION **m**
LOCATION **ar**
ENEMY-ORIENTATION **av**
ENGAGEMENT-AREA **e**

Verify
plausible upper bound
(company **c** (COMMANDS **P2T**) (COMMANDS **PI**) (COMMANDS **P1T**))
(overlay-object **ar)**
(overlay-object **av** (PART-OF **ar)**)
(overlay-object **e** (PART-OF **av)**)
(mission **m** (WITH **c**) (IN **ar)**)

plausible lower bound
(company-D **c** (COMMANDS **P2T**) (COMMANDS **PI**) (COMMANDS **P1T**))
(company-D-area-of-responsibility **ar)**
(avenue-of-approach-D **av** (PART-OF **ar)**)
(engagement-area-D **e** (PART-OF **av)**)
(defend-area-mission-D **m** (WITH **c**) (IN **ar)**)

Find
plausible upper bound
(platoon **PI** (WEAPONS-CLASSIFICATION "light"))
(platoon **P1T**)
(platoon **P2T**)
(hill-sector **hsi** (IN **ar)** (VISIBLE **e**))
(hill-sector **hs1tp** (IN **ar)** (VISIBLE **e**))
(hill-sector **hs2tp** (IN **ar)** (VISIBLE **e**))

plausible lower bound
(platoon-D3 **PI** (WEAPONS-CLASSIFICATION "light"))
(platoon-D1 **P1T**)
(platoon-D2 **P2T**)
(hill-sector-868-1 **hsi** (IN **ar)** (VISIBLE **e**))
(hill-sector-863-2 **hs1tp** (IN **ar)** (VISIBLE **e**))
(hill-sector-875-2D **hs2tp** (IN **ar)** (VISIBLE **e**))

Perform
POSITION-PLATOON UNIT-ID **PI**
LOCATION **hsi**
POSITION-PLATOON UNIT-ID **P1T**
LOCATION **hs1tp**
POSITION-PLATOON UNIT-ID **P2T**
LOCATION **hs2tp**

Fig. 9. Initial Procedure Formed by Captain.

The plausible lower bound is the conjunction of the selected explanations, re-expressed in terms of the variables from the procedure's conclusion. In other words, the plausible lower

bound covers only the initial example from Fig. 7. The plausible upper bound is an over-generalization of the plausible lower bound, in which individual objects are turned into the more general objects (the heuristic used is to take the individual's ancestor one down in the generalization hierarchy from the top level of the hierarchy) and the relationships between the objects are preserved.

Experimentation

The agent will use the plausible version space in Fig. 9 to generate other placements for defensive missions, and will show these to the user, who will accept or reject them. The user can control this experimentation process by fixing some of the parameters of the defensive mission. For instance, it is useful to ask the agent to initially generate only placements of Company D in its area of responsibility as shown below, using a user-guided search method (Hieb, 1996). This limits the search space the agent must deal with.

User Settings for Experimentation Menu:

1> Make Variables Distinct
2> Fix Variables
3> Continue

Enter Selection Number: 2

Enter one or a list of variables to fix (or q to quit):
(c, e)

fixed variable **c** to company-D
fixed variable **e** to engagement-area-D

The Captain agent generates a new placement of Company D by generating placements consistent with the plausible upper bound of the procedure in Fig. 9. Placements that fall under the lower bound as well are discarded. It then proposes a placement (which is covered by the upper bound but not the lower bound) to the user on the ModSAF screen, as shown in Figures 10 and 11. The user rejects this placement. The system then asks that the user look at the initial placement to determine what additional explanations need to be given to correct the initial procedure such that it will not generate any more incorrect examples[3]. The user then explains that the infantry unit is too far away from the area of engagement, as shown in Appendix C.

[3] The initial positive example is always used as a reference, so that the user can generate a *positive* explanation, which will correct the current procedure. Explanations of why the negative example is incorrect can also be generated, but currently cannot be used to modify the plausible version space. Such *negative* explanations would have to be converted to *positive* explanations. An additional benefit to referring back to the initial example is that the user is quite familiar with it, as the user specified it at the start of the learning process as a prototypical solution.

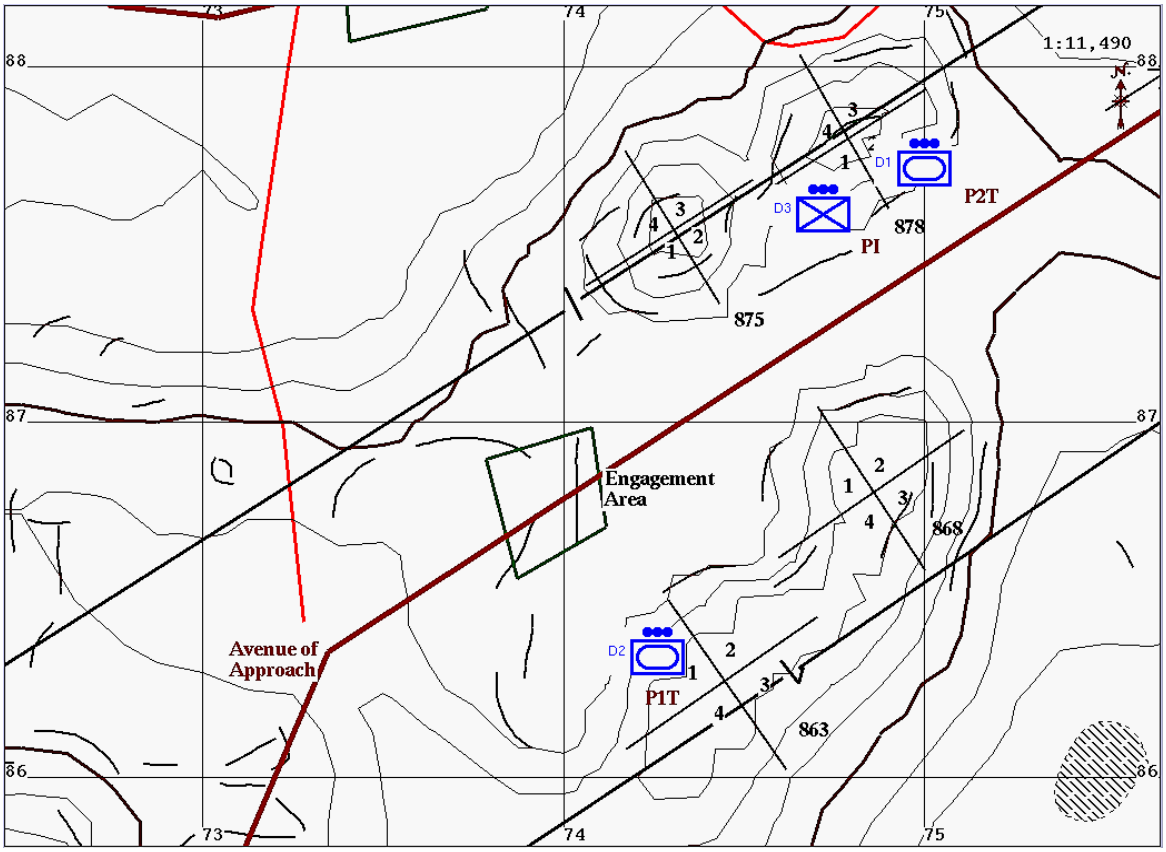

Fig. 10. Negative Company D Placement Generated by Captain.

to accomplish:

	POSITION-COMPANY	UNIT-ID (company-D **c**)
		MISSION (defend-area-mission-D **m)**
		LOCATION (company-D-area-of-responsibility **ar)**
		ENEMY-ORIENTATION (avenue-of-approach-D **av)**
		ENGAGEMENT-AREA (engagement-area-D **e**)
perform:		
	POSITION-PLATOON	UNIT-ID (platoon-D3 **PI**)
		LOCATION (hill-sector-878-1 **hsi**)
	POSITION-PLATOON	UNIT-ID (platoon-D2 **P1T**)
		LOCATION (hill-sector-863-1 **hs1tp**)
	POSITION-PLATOON	UNIT-ID (platoon-D1 **P2T**)
		LOCATION (hill-sector-878-2 **hs2tp**)

Fig. 11. Negative Company D Problem / Solution.

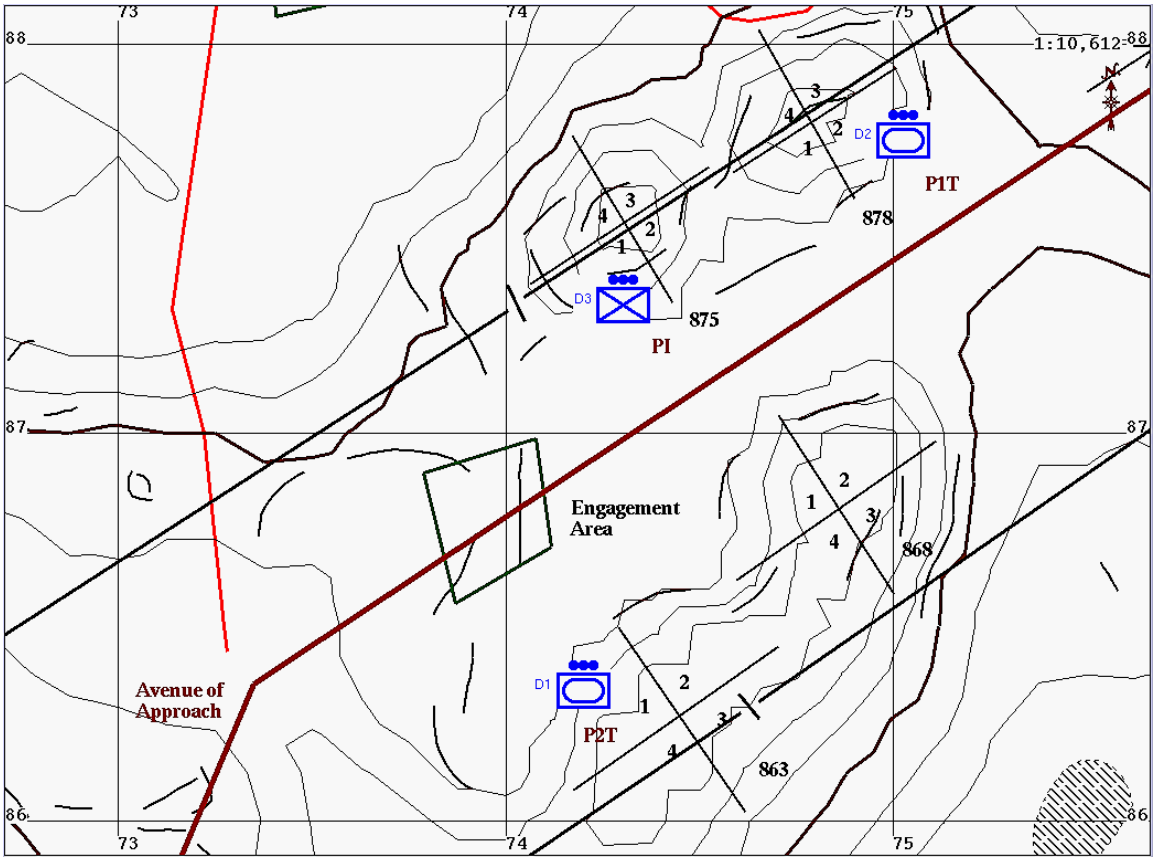

Fig. 12. Positive Company D Placement Generated by Captain.

to accomplish:

POSITION-COMPANY	UNIT-ID (company-D **c**)
	MISSION (defend-area-mission-D **m)**
	LOCATION (company-D-area-of-responsibility **ar)**
	ENEMY-ORIENTATION (avenue-of-approach-D **av)**
	ENGAGEMENT-AREA (engagement-area-D **e**)

perform:

POSITION-PLATOON	UNIT-ID (platoon-D3 **PI**)
	LOCATION (hill-sector-875-1 **hsi**)
POSITION-PLATOON	UNIT-ID (platoon-D2 **P1T**)
	LOCATION (hill-sector-878-2 **hs1tp**)
POSITION-PLATOON	UNIT-ID (platoon-D1 **P2T**)
	LOCATION (hill-sector-863-1 **hs2tp**)

Fig. 13. Positive Company D Problem / Solution

At this point Captain generates the placement in Fig. 12, which the expert subsequently accepts. Consequently, the system makes the following generalizations in the lower bound that correspond to the generalization of the positive examples from Fig. 7 and Fig. 12:

Is this correct solution to the problem:?
[y/n]: y

I made the following generalization(s):
hill-sector-868-1 hill-sector-863-1 → hill-sector
hill-sector-863-2 hill-sector-878-2 → hill-sector
hill-sector-875-2 hill-sector-863-2 → hill-sector
platoon-D1 platoon-D2 → armored-platoon
platoon-D2 platoon-D1 → armored-platoon

At this point, all the placement examples for Company-D that the system might generate are already covered by the plausible lower bound of the version space and Captain presents the following message to the user:

There are no more examples from which to learn
2340 solutions of the current problem were examined
335 solutions are available for verification

Would you like to see other problem solving examples to verify the learned procedure? [y/n]: n

The messages indicate that during the learning process, 2340 tuples from the initial search space have been examined. Of the 2340 tuples, 2005 tuples were not covered by the upper bound, and 335 tuples are covered by the lower bound. The tuples covered by the lower bound are considered positive examples of the placement problem for Company D and are saved. The user has the option to review these positive placements in the same manner as during the experimentation above. If the user finds an incorrect placement, the rule is modified accordingly.

The user then directs Captain to experiment with placing other companies for defending their areas of responsibility by "unfixing" the previously-fixed objects, company-D and engagement-area-D[4]. In response, the system generates a new example for the user to validate. This example comes from a different area on the map, which is in the area of responsibility for Company E.

Because the user accepts the placement generated by the agent for Company E, the system is able to make a significant reduction in the plausible version space by generalizing the following concepts from the plausible lower bound:

Is this correct solution to the problem:?
[y/n]: y
I made the following generalization(s):

company-D company-E → company
avenue-of-approach-D avenue-of-approach-E → avenue-of-approach

[4] This involves saving the procedure and then loading it again to generate a new search space without any fixed variables.

engagement-area-D engagement-area-E → engagement-area
defend-area-mission-D defend-area-mission-E → Defend-area-mission
platoon-D3 platoon-E3 → infantry-platoon
company-D-area-of-responsibility company-E-area-of-responsibility → area-of-responsibility

The process described above is followed until Captain is able to place companies successfully on the first try. Typically each placement is dealt with in fewer examples and at the same time learning the values of all possible parameters, for example by varying the value of the engagement-area that was previously fixed. In the end the agent has learned the procedure shown in Fig. 14. It produces the final message:

There are no more examples from which to learn
2644 solutions of the current problem were examined
375 solutions are available for verification

Would you like to see other problem solving examples to verify the learned procedure? [y/n]: n

The messages indicate that during the learning process, 2644 tuples from the initial search space have been examined. Of the 2644 tuples, 2269 tuples were not covered by the upper bound, and 375 tuples are covered by the lower bound. The tuples covered by the lower bound are considered positive examples of the placement problem and are saved.

Applying the Learned Procedure

Subsequently, after the procedure has been learned for the defensive placement mission, another company commander may use the procedure. A scenario is shown in Fig. 19 for Company A where the company command agent has been ordered to defend the area shown on the map.

The agent will look for a procedure that can be used for this type of mission – and finds the procedure shown in Fig. 14 Next it will check if the verify portion of the lower bound condition of this procedure is satisfied when the variables from the procedure are matched with the objects from the specific mission. Using the procedure in Fig. 14 it will check if company-A (that matched **c**) is a company that commands three platoons A1, A2 and A3. This matching succeeds because company-A COMMANDS platoon-A1, platoon-A2 and platoon-A3 and company-A has 3 platoons. Similarly the agent will check if company-A-area-of-responsibility, engagement-area-A and defend-area-mission-A satisfy the corresponding conditions from the verify portion of the plausible lower bound (i.e. the conditions for **ar**, **e** and **m** respectively). Since the verify portion of the lower bound matched successfully, the solve portion of the lower bound is used to find objects which correctly satisfy the constraints of the associated clauses. A1 and A2 must be armored-platoons, and A3 must be an infantry-platoon with WEAPON-CLASSIFICATION "light", such as platoon-A3.

To Accomplish
POSITION-COMPANY UNIT-ID **c**
MISSION **m**
LOCATION **ar**
ENEMY-ORIENTATION **av**
ENGAGEMENT-AREA **e**

Verify
plausible upper bound
(company **c** (NUMBER-OF-PLATOONS 3)(COMMANDS **P2T**)
(COMMANDS **Pl**) (COMMANDS **P1T**))
(overlay-object **ar)**
(overlay-object **av** (PART-OF **ar)**)
(overlay-object **e** (PART-OF **av)**)
(mission **m** (WITH **c**) (IN **ar)**)

plausible lower bound
(company **c** (NUMBER-OF-PLATOONS 3)(COMMANDS **P2T**)
(COMMANDS **Pl**) (COMMANDS **P1T**))
(area-of-responsibility **ar)**
(mounted-avenue-of-approach **av** (PART-OF **ar)**)
(engagement-area **e** (PART-OF **av)**)
(defend-area-mission **m** (WITH **c**) (IN **ar)**)

Solve
plausible upper bound
(platoon **Pl** (WEAPONS-CLASSIFICATION "light"))
(platoon **P1T**)
(platoon **P2T**)
(hill-sector **hsi** (IN **ar)** (VISIBLE **e**) (DISTANCE-TO-ENGAGEMENT-AREA "close"))
(hill-sector **hs1tp** (IN **ar)** (VISIBLE **e**))
(hill-sector **hs2tp** (IN **ar)** (VISIBLE **e**))

plausible lower bound
(infantry-platoon **Pl** (WEAPONS-CLASSIFICATION "light"))
(armored-platoon **P1T**)
(armored-platoon **P2T**)
(hill-sector **hsi** (IN **ar)** (VISIBLE **e**) (DISTANCE-TO-ENGAGEMENT-AREA "close"))
(hill-sector **hs1tp** (IN **ar)** (VISIBLE **e**))
(hill-sector **hs2tp** (iN **ar)** (VISIBLE **e**))

Perform
POSITION-PLATOON UNIT-ID **Pl**
LOCATION **hsi**
POSITION-PLATOON UNIT-ID **P1T**
LOCATION **hs1tp**
POSITION-PLATOON UNIT-ID **P2T**
LOCATION **hs2tp**

Fig. 14. Final Procedure Formed by Captain.

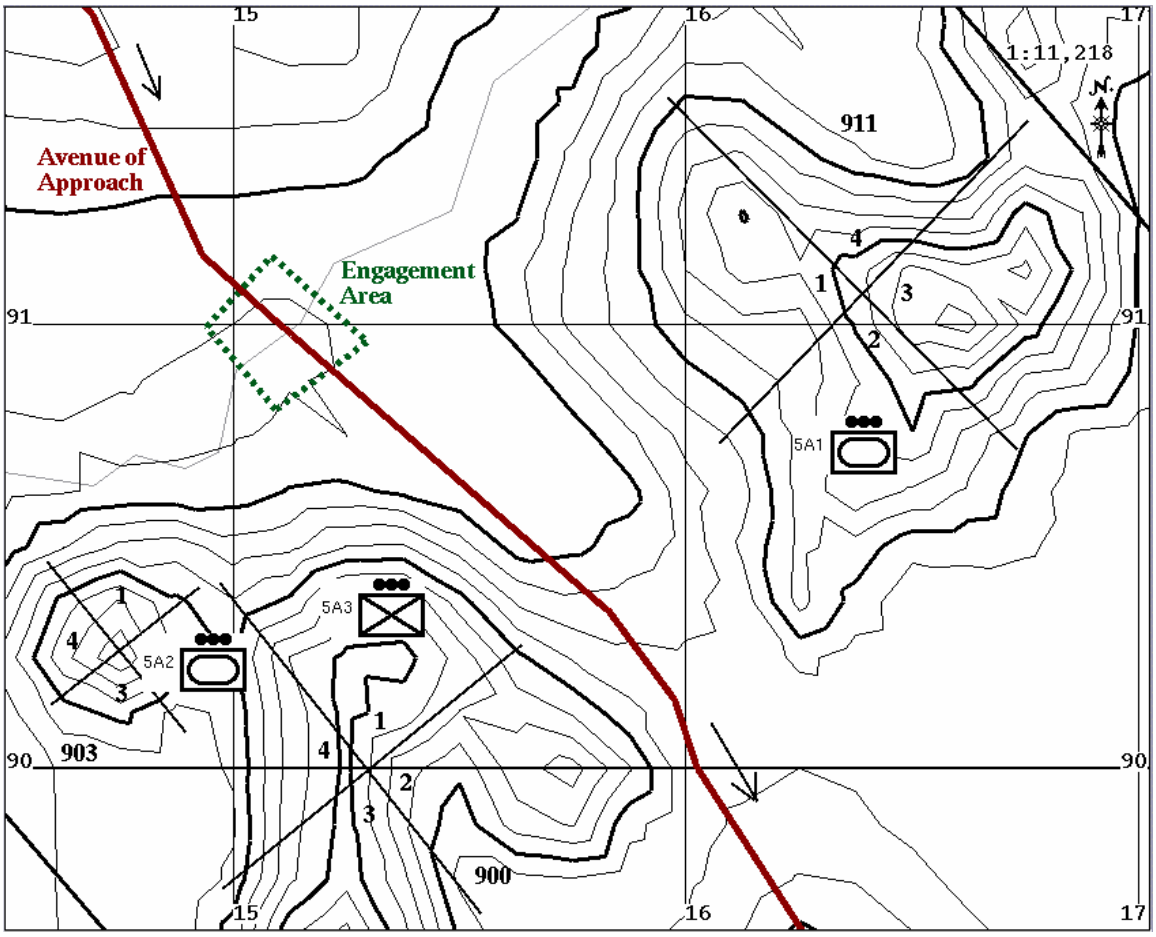

Fig. 15. Placement of Platoons by a Captain Agent for a Defensive Mission.

The rest of the variables from the solve portion (i.e. **hs1tp**, **hs2tp**, **hsi**) have to be matched with objects from the map. For instance, **hsi** may be matched with hill-sector-900-1 because this hill sector is IN company-A-area-of-responsibility, the engagement-area-A is VISIBLE from it, and the DISTANCE-TO-ENGAGEMENT-AREA is "close". Similarly, **hs1tp** and **hs2tp** could be successfully matched to hill-sector-911-2 and hill-sector-903-2. Because the objects have been found to satisfy the solve portion of the lower bound condition, the procedure is applied and the solution indicated is shown in Fig. 15 and represented as:

POSITION-PLATOON	platoon-A3	IN	hill-sector-900-1
POSITION-PLATOON	platoon-A1	IN	hill-sector-911-2
POSITION-PLATOON	platoon-A2	IN	hill-sector-903-2

It is important to stress that there are many correct placements, corresponding to different ways of matching the lower bound to the situation the agent faces. For instance, **hs1tp**, **hs2tp**, **hsi** could also be successfully matched with another set of hill sectors that satisfy the lower bound conditions of the procedure leading to the following solution of the placement problem:

POSITION-PLATOON	platoon-A3	IN	hill-sector-903-1
POSITION-PLATOON	platoon-A1	IN	hill-sector-903-2
POSITION-PLATOON	platoon-A2	IN	hill-sector-911-1

If the lower bound does not mach the current situation, the procedure is not considered applicable. Another problem-solving strategy could be that if the lower bound of the procedure does not match the current situation, then the agent would check if the upper bound matches it, and if it does, then the procedure is again applied, but the unit placement indicated by it is considered only plausibly correct.

V. ANALYSIS OF LEARNING

It is important to stress that while this procedure has been learned from six examples, the agent internally examined approximately 5,000 different placements that are covered by the upper bound of the procedures in Figures 9 and 14. These placements are for the three areas considered corresponding to Companies D, E and F. The learning process stopped because the procedure was refined to where all the placements that could be generated were covered by the lower bound of the procedure being learned (there was no other placement both covered by the upper bound and not covered by the lower bound). It is would be impractical for the ModSAF user to consider each of the 5,000 placements individually. However, the user may continue to verify the learned procedure by examining placements covered by the plausible lower bound (there are approximately 700 such placements in the training scenario illustrated).

The initial search space for the problem without any explanations given was approximately 4×10^{13}. The procedure in Fig. 14 was learned from 3 positive and 3 negative examples – one given by the user and 5 generated for classification by Captain. 17 explanations were given by the user, 14 initially, and 3 during experimentation. If fewer initial explanations were given to Captain, more examples would be generated for user classification.

This illustration gives a general outline of the learning method. There are many other kinds of interactions between the user and the agent. For instance, the user may choose to give the agent additional examples of good placements. These may cause the generalization of the lower bound or of both the lower and the upper bounds.

During learning, the agent may also accumulate negative or positive exceptions of the procedure. These are incorrect placements that are covered by the lower bound, or good placements not covered by the upper bound. In such cases, the agent will attempt to elicit new knowledge (e.g. new features of platoons or their positions that are not defined in the knowledge base) from the user using the consistency-driven elicitation methods detailed in

[18]. These knowledge items will allow the agent to modify the plausible version space of the procedure such that the negative and the positive exceptions become negative examples and positive examples, respectively. Another way of dealing with a procedure's positive exceptions is to split the plausible version space into several plausible version spaces that do not have exceptions [4]. This will lead to learning more smaller procedures for the particular problem (instead of one large procedure) that correspond to smaller plausible version spaces, giving a more accurate solution

The general idea of this approach is to allow the user to teach the agent in a variety of ways, and to intervene whenever the user wishes in the teaching process. On the other hand, the agent learner has a very proactive strategy of soliciting explanations in a variety of ways in order to remedy its failures. Because this approach is based on a user interacting with, checking and correcting the way the agents solve problems, it produces verified knowledge-based agents without an initial verification step.

Based on experiments with Captain, we believe it can learn defensive placement techniques for any well-documented terrain, involving larger than company forces. As the previous section showed, complex defensive placement procedures are learned from a small number of examples. This ability to learn with only a few examples is due to explanations that identify the relevant features of the examples. In order to quantify the value of explanations we ran a series of experiments. These experiments were a variation of the leave-one-out experiments done with empirical inductive learners. First, an ideal procedure was learned for a certain problem in the ModSAF domain by giving 8 explanations. Then, a series of procedures was learned by withholding explanations from the system in a progressive fashion (giving first one less, than two less and so on). We also varied the withheld explanations to generate all possible combinations of the remaining explanations. Thus by leaving out 1 explanation, we had 8 combinations, by leaving out 2 explanations we had 28 combinations, and so on. The initial search space (which in the Disciple approach is approximated by the set of instances of the plausible upper bound) was measured for each procedure to determine what the effect of leaving out the explanations was. In all we performed about 40 experiments to generate this graph. In each case Captain formed a plausible version space rule that could be used to solve the problem.

The graph in Fig. 16 was obtained by averaging the search space obtained for each number of explanations given We found a wide variation of the search space since the individual utility of the explanations varied. Both the minimum and maximum values of the search space at each number of explanations are also presented on the graph to show this variation. Fig. 16 shows that there is roughly an order of magnitude drop in the search space that the algorithm uses for each explanation given. The maximum search space is 9.54×10^8. The initial explanations vastly

reduce this search space. As an example, after two explanations are given, it averages 3.1 x 10^6 placements. This result explains why the user can teach the agent the procedure after seeing only a small number of examples.

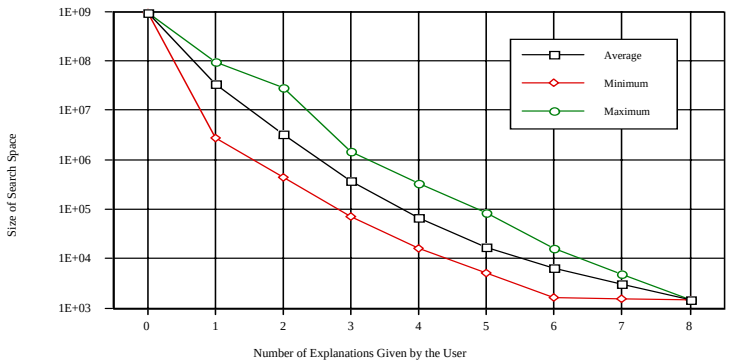

Fig. 16. Reduction of Search Space by Explanations

VI. CONCLUSIONS

Systems for automating complex tasks must be designed so that they can be general enough to be adapted to different domains. For example, considerable effort was expended in modifying both the ModSAF application (which contains over 450 source libraries written in C) and Agent-Disciple to create Captain [21]. A future goal in Agent-Disciple is to use the existing editor interface of ModSAF, as opposed to a separate learning system interface. Lieberman [12] points out that the interface between an end user and the agent training system is a crucial issue not addressed in most of the machine learning research. The Agent-Disciple approach is to use as much of the existing ModSAF interface as possible, on the assumption that this is easier for the SME.

Much of the power of the agent instruction approach presented comes from the multiple types of interaction between the SME and the agent being taught. Such rich interaction is rare in Machine Learning systems, and is closer to the interaction found in Programming By Demonstration systems [13]. Such interaction is necessary, however, to develop more powerful agents. These interactions include: specifying new terms in the representation language of the agent; giving the agent an example of a solution to a task for which the agent is to learn a general procedure; validating analogical instances of solutions proposed by the agent; explaining to the agent reasons for the validation; and being guided to provide new terms in the representation during interaction [18].

Captain addresses the basic requirements for an ideal Programming By Demonstration learner, as identified by Maulsby and Witten [14]. First, the learning agent is under the user's

control, who specifies the actions and features relevant to the task to be taught, gives hints and explanations to the agent, and guides its learning actions. Second, the learning agent uses various knowledge-based heuristics for performing plausible generalizations and specializations that are understandable, including plausible generalization of a single example. Third, the agent learns a task in terms of all the parameters necessary for task execution. It also learns from a small set of examples.

Captain currently does not address autonomous learning, where the agent would learn without the guidance of an SME, but the same learning methods that are being developed for instruction should be applicable [7], [8], [9], [19].

Verification and validation is a difficult problem with military command agents, because of the complexity of the agent reasoning process. Our approach addresses this problem by allowing the user to test the agent with additional examples after the agent has successfully learned how to perform a mission. The SME can select the testing examples or the testing examples can be automatically generated. If the agent performs the mission incorrectly, the user can correct the agent through the same instruction techniques that were originally used to teach the agent (i.e., by giving additional examples or explanations). If the agent performs the mission selected by the SME correctly, then confidence in the learned behavior increases.

Captain offers an efficient approach for teaching complex behavior to an agent through demonstration. This approach was illustrated by our investigations with the Captain system. This approach to training ModSAF agents appears to be more natural and significantly simpler than the currently process, where the SME manually specifies the mission of the ModSAF agents in great detail to achieve reasonable behavior in a simulation. The learning efficiency in Captain is achieved through the use of plausible version spaces and a human guided heuristic search of these spaces.

ACKNOWLEDGEMENTS

This work would not have been possible without the Disciple system of Gheorghe Tecuci. The authors thank Ken Frosch of the GMU C3I Center for designing/implementing the Disciple/ModSAF PDU interface, and Vince Laviano of the GMU C3I Center for ModSAF programming.

REFERENCES

[1] R.A. Brooks, “The Whole Iguana,” in *Robotic Science*, M. Brady, Ed. Cambridge, MA: MIT Press, pp. 432-456, 1989.

[2] A. Cypher, Ed., *Watch What I Do: Programming by Demonstration*, Cambridge, MA: MIT Press, 1993.

[3] T. Dybala and G. Tecuci, "Shared Expertise Space: A Learning Oriented Model for Cooperative Engineering Design," *Proc. IJCAI-95 Workshop on Machine Learning in Engineering*, Montreal, Canada, August, 1995.

[4] M.R. Hieb, *Training Instructable Agents Through Plausible Version Space Learning*, PhD Dissertation, School of Information Technology and Engineering, George Mason University, Fairfax, VA, 1996.

[5] M.R. Hieb and G. Tecuci, "Training an Agent through Demonstration: A Plausible Version Spaces Approach" *Proc. of the 1996 AAAI Spring Symposium on Acquisition, Learning and Demonstration: Automating Tasks for Users.* AAAI Press Technical Report, Menlo Park, CA, 1996.

[6] M.R. Hieb, G. Tecuci, J.M. Pullen, A. Ceranowicz, and D. Hille, "A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces," in *Proc. of the 5th Conference on Computer Generated Forces and Behavioral Representation*., Orlando, Florida, 1995.

[7] M.R. Hieb, D. Hille and G. Tecuci, "Designing a Computer Opponent for War Games: Integrating Planning, Learning and Knowledge Acquisition in WARGLES," *Proce. of the 1993 AAAI Fall Symposium on Games: Learning and Planning*, AAAI Press Technical Report FS-93-02, Menlo Park, CA, 1993.

[8] D. Hille, M.R. Hieb, J.M. Pullen and G. Tecuci, "Abstracting Terrain Data through Semantic Terrain Transformations," *Proc. of the 5th Conference on Computer Generated Forces and Behavioral Representation*, Orlando, Florida, 1995.

[9] D. Hille, M.R. Hieb and G. Tecuci, "Captain: Building Agents that Plan and Learn," *Proc. of the 4th Conference on Computer Generated Forces and Behavioral Representation*., Orlando, Florida, 1994.

[10] S.B. Huffman, *Instructable Autonomous Agents*. PhD Dissertation, Department of Computer Science and Engineering, University of Michigan, 1994.

[11] J.E. Laird, A. Newell and P.S. Rosenbloom, "Soar: An Architecture for General Intelligence," *Artificial Intelligence*, Vol. 33, 1987.

[12] H. Lieberman, "A User Interface for Knowledge Acquisition From Video," In *Proc. Eleventh Conference on Artificial Intelligence*, San Francisco, CA: Morgan Kaufmann, 1994.

[13] D. Maulsby, *Instructable Agents*, PhD Dissertation, Department of Computer Science, University of Calgary, 1994.

[14] D. Maulsby, and I.H. Witten, "Learning to Describe Data in Actions," *Proc of ICML-95 Workshop on Learning from Examples vs. Programming by Demonstration*, CA, 1995.

[15] M. Tambe, W.L. Johnson, R.M. Jones, F. Koss, J.E. Laird, P.S. Rosenbloom, and K. Schwamb, "Intelligent Agents for Interactive Simulation Environments," *AI Magazine.*, 16(1), Spring, 1995.

[16] G. Tecuci, *DISCIPLE: A Theory, Methodology and System for Learning Expert Knowledge*, Ph.D. Dissertation, University of Paris South, 1988.

[17] G. Tecuci, "Automating Knowledge Acquisition as Extending, Updating and Improving a Knowledge Base," *IEEE Trans. of Syst. Man Cybern.*, vol. SMC-22(6), 1992.

[18] G. Tecuci and M.R. Hieb, "Consistency-driven Knowledge Elicitation: Using a Machine Learning-oriented Knowledge Representation to Integrate Learning and Knowledge Elicitation in NeoDISCIPLE," *Knowledge Acquisition*, vol. 6(1), 1994.

[19] G. Tecuci, Hieb M.R., D. Hille and J.M. Pullen, "Building Adaptive Autonomous Agents for Adversarial Domains," *Proc. of the AAAI 94 Fall Symposium – Planning and Learning: On To Real Applications.*, 1994.

[20] G. Tecuci and Y. Kodratoff, "Apprenticeship Learning in Imperfect Theory Domains" in *Machine Learning: An Artificial Intelligence Approach.* Vol. III, Y. Kodratoff and R. S. Michalski, Eds., San Francisco, CA: Morgan Kaufmann, 1990.

[21] E.L. White, K.E. Frosch, V.P. Laviano, M.R. Hieb, and J.M. Pullen, "Interfacing External Decision Processes to DIS Applications," *8th Conference on Computer Generated Forces and Behavioral Representation*., 1996.

AUTHORS' BIOGRAPHIES

Michael Hieb received his PhD in Information Technology at George Mason University in 1996. Dr. Hieb is currently working for AB Technologies. He has published over 30 papers in the areas of learning agents, knowledge acquisition and multistrategy learning. When working for IntelliTek, Dr. Hieb implemented a distributed problem-solving testbed at the Goddard Space Flight Center. Previously, he worked as a Nuclear Engineer for General Electric.

J. Mark Pullen is Associate Professor of Computer Science at George Mason University. He also is a member of the Center for Excellence in Command, Control, Communications and Intelligence. He was an officer in the U.S. Army for 21 years, specializing in advanced computer and communications technologies. Dr. Pullen was with the Defense Advanced Research Projects Agency (DARPA) from 1986 to 1992, where he was Program Manager for Advanced Computing, Networking and Distributed Simulation, and Deputy Director of the Tactical Technology Office and the Information Science and Technology Office. His research interests include distributed and parallel computing systems and their applications to educational and military simulations. He is a Fellow of the IEEE.

Appendix A
Glossary of Military Terms

area of responsibility: area established by boundary lines, within which a unit is expected to operate.

armored: attribute indicating use of vehicles with a defensive covering, usually tanks or attack: application of combat power on an objective by a military force in order to a make the enemy abandon their defense or face piecemeal destruction.

avenue of approach: an air or ground route of an attacking force of a given size leading to its objective or to key terrain in its path.

company: unit usually consisting of 3 or 4 platoons.

control measure: area of responsibility, avenue of approach, engagement area, or other method whereby a commander directs activity as part of a mission given to a subordinate unit.

defend: actions taken by a military force to cause an enemy attack to fail; usually focused on retaining control of terrain.

engagement area: location where a unit attacks or defends.

infantry: unit consisting of foot soldiers.

march: normal forward movement of a unit (including vehicles as well as units on foot).

mobility corridor: area within which a unit can move at its normal march rate.

ModSAF: Modular Semi-Automated Forces.

orientation: direction something is facing.

overlay: information superimposed on a map, chart, or other display, to show details not appearing or requiring special emphasis on the original placement: area where a military unit stays for some length of time; usually the intention is to defend the area

platoon: unit of about 40 humans or 4 tanks.

quadrant: one quarter of a placement or other area.

SAFOR: Semi-Automated Forces.

sector: an area designated by boundaries within which a unit operates, and for which it is responsible.

unit: small military force under a single commander.

Appendix B
First Phase of Solution Explanation by Expert

Explanation Editor

Indicate - a variable that you would like to generate explanations from
- a pair of variables to generate explanations from
- an explanation (enter ? for help with syntax)
- 'all' to generate all explanations
- 'retract' to retract an explanation
- 'view' to view the current set of accepted explanations
- 'quit' to quit explanation generation

[variable/variable pair/explanation/all/r/v/q]: c

Choose the relevant explanation(s)

1> (**c** IS-A company-D AFFILIATION "friendly")
2> (**c** IS-A company-D NUMBER-OF-PLATOONS 3)
3> (**m** IS-A defend-area-mission-D WITH **c** IS-A company-D)
4> (**c** IS-A company-D COMMANDS **PI** IS-A platoon-D3)
5> (**c** IS-A company-D COMMANDS **P1T** IS-A platoon-D1)
6> (**c** IS-A company-D COMMANDS **P2T** IS-A platoon-D2)
7> QUIT

Enter selection as number or list of numbers: (3 4 5 6)
[variable/variable pair/explanation/all/r/v/q]: m
Choose the relevant explanation(s)

1> (**m** IS-A defend-area-mission-D IN **ar** IS-A company-D-area-of-responsibility)
2> QUIT

Enter selection as number or list of numbers: 1
[variable/variable pair/explanation/all/r/v/q]: ar

Choose the relevant explanation(s)

1> (**av** IS-A avenue-of-approach-D PART-OF **ar** IS-A company-D-area-of-responsibility)
2> (**hsi** IS-A hill-sector-868-1 IN **ar** IS-A company-D-area-of-responsibility)
3> (**hs1tp** IS-A hill-sector-863-2 IN **ar** IS-A company-D-area-of-responsibility)
4> (**hs2tp** IS-A hill-sector-875-2D IN **ar** IS-A company-D-area-of-responsibility)
5> QUIT

Enter selection as number or list of numbers: (1 2 3 4)
[variable/variable pair/explanation/all/r/v/q]: e

Choose the relevant explanation(s)
1> (**e** IS-A engagement-area-D WIDTH 0.17)
2> (**e** IS-A engagement-area-D LENGTH 1.3)
3> (**e** IS-A engagement-area-D PART-OF **av** IS-A avenue-of-approach-D)
4> (**hsi** IS-A hill-sector-868-1 VISIBLE **e** IS-A engagement-area-D)
5> (**hs1tp** IS-A hill-sector-863-2 VISIBLE **e** IS-A engagement-area-D)
6> (**hs2tp** IS-A hill-sector-875-2D VISIBLE **e** IS-A engagement-area-D)
7> QUIT

Enter selection as number or list of numbers: (3 4 5 6)
[variable/variable pair/explanation/all/r/v/q]: v

Accepted Explanations:
1> (**c** IS-A company-D COMMANDS **P2T** IS-A platoon-D2)
2> (**c** IS-A company-D COMMANDS **P1T** IS-A platoon-D1)
3> (**c** IS-A company-D COMMANDS **PI** IS-A platoon-D3)
4> (**m** IS-A defend-area-mission-D WITH **c** IS-A company-D)
5> (**m** IS-A defend-area-mission-D IN **ar** IS-A company-D-area-of-responsibility)
6> (**hs2tp** IS-A hill-sector-875-2D IN **ar** IS-A company-D-area-of-responsibility)
7> (**hs1tp** IS-A hill-sector-863-2 IN **ar** IS-A company-D-area-of-responsibility)
8> (**hsi** IS-A hill-sector-868-1 IN **ar** IS-A company-D-area-of-responsibility)
9> (**av** IS-A avenue-of-approach-D PART-OF **ar** IS-A company-D-area-of-responsibility)
10> (**hs2tp** IS-A hill-sector-875-2D VISIBLE **e** IS-A engagement-area-D)
11> (**hs1tp** IS-A hill-sector-863-2 VISIBLE **e** IS-A engagement-area-D)
12> (**hsi** IS-A hill-sector-868-1 VISIBLE **e** IS-A engagement-area-D)
13> (**e** IS-A engagement-area-D PART-OF **av** IS-A avenue-of-approach-D)

[variable/variable pair/explanation/all/r/v/q]: `pi`

Choose the relevant explanation(s)

1> (**PI** IS-A platoon-D3 MARKING "D3")
2> (**PI** IS-A platoon-D3 WEAPONS-CLASSIFICATION "light")
3> (**PI** IS-A platoon-D3 MODSAF-UNIT-NAME "unit_US_DIGroup_Platoon")
4> (**PI** IS-A platoon-D3 EFFECTIVE-RANGE "close")
5> (**hsi** IS-A hill-sector-868-1 DISTANCE-TO-ENGAGEMENT-AREA "close")
(**PI** IS-A platoon-D3 EFFECTIVE-RANGE "close")
6> (**P1T** IS-A platoon-D1 EFFECTIVE-RANGE "close")
(**PI** IS-A platoon-D3 EFFECTIVE-RANGE "close")
7> (**hs1tp** IS-A hill-sector-863-2 DISTANCE-TO-ENGAGEMENT-AREA "close")
(**PI** IS-A platoon-D3 EFFECTIVE-RANGE "close")
8> (**P2T** IS-A platoon-D2 EFFECTIVE-RANGE "close")
(**PI** IS-A platoon-D3 EFFECTIVE-RANGE "close")
9> QUIT

Enter selection as number or list of numbers: `2`

[variable/variable pair/explanation/all/r/v/q]: `q`

Appendix C

Second Phase of Solution Explanation by Expert

Is this correct solution to the problem:?
[y/n]: n

Look again at the initial problem solving example you specified

1> Explanation Editor
2> Blame Object
3> Specialization by System
4> View Initial Example
5> View Current Example
6> Quit

Enter Selection Number: 1
[variable/variable pair/explanation/all/r/v/q]: hsi

Choose the relevant explanation(s)

1> (**hsi** IS-A hill-sector-868-1 VISIBLE mobility-corridor-D PART-OF **av** IS-A avenue-of-approach-D)
2> (**PI** IS-A platoon-D3 EFFECTIVE-RANGE "close")
(**hsi** IS-A hill-sector-868-1 DISTANCE-TO-ENGAGEMENT-AREA "close")
3> (**P1T** IS-A platoon-D1 EFFECTIVE-RANGE "close")
(**hsi** IS-A hill-sector-868-1 DISTANCE-TO-ENGAGEMENT-AREA "close")
4> (**hs1tp** IS-A hill-sector-863-2 VISIBLE mobility-corridor-D)
(**hsi** IS-A hill-sector-868-1 VISIBLE mobility-corridor-D)
5> (**hs1tp** IS-A hill-sector-863-2 DISTANCE-TO-ENGAGEMENT-AREA "close")
(**hsi** IS-A hill-sector-868-1 DISTANCE-TO-ENGAGEMENT-AREA "close")
6> (**hs1tp** IS-A hill-sector-863-2 HAS-EXIT "yes")
(**hsi** IS-A hill-sector-868-1 HAS-EXIT "yes")
7> (**P2T** IS-A platoon-D2 EFFECTIVE-RANGE "close")
(**hsi** IS-A hill-sector-868-1 DISTANCE-TO-ENGAGEMENT-AREA "close")
8> (**hs2tp** IS-A hill-sector-875-2D HAS-EXIT "yes")
(**hsi** IS-A hill-sector-868-1 HAS-EXIT "yes")
9> CONTINUE
10> QUIT

Enter selection as number or list of numbers: 9

Choose the relevant explanation(s)

1> (**hsi** IS-A hill-sector-868-1 PART-OF HILL868)
2> (**hsi** IS-A hill-sector-868-1 DISTANCE-TO-ENGAGEMENT-AREA "close")
3> (**hsi** IS-A hill-sector-868-1 HAS-EXIT "yes")
4> (**hsi** IS-A hill-sector-868-1 VISIBLE mobility-corridor-D)
5> (**hsi** IS-A hill-sector-868-1 QUADRANT 1)
6> QUIT

Enter selection as number or list of numbers: 2

As a result of this explanation, the property value pair (DISTANCE-TO-ENGAGEMENT-AREA "close") is added to the clause for the variable **hsi** in both the upper and lower bound of the procedure in Fig. 9, as shown below:

old plausible upper bound
(hill-sector **hsi** (IN **ar)** (VISIBLE **e**))

new plausible upper bound
(hill-sector **hsi** (IN **ar)** (VISIBLE **e**) (DISTANCE-TO-ENGAGEMENT-AREA "close"))

old plausible lower bound
(hill-sector-868-1 **hsi** (IN **ar)** (VISIBLE **e**))

new plausible lower bound
(hill-sector-868-1 **hsi** (IN **ar)** (VISIBLE **e**) (DISTANCE-TO-ENGAGEMENT-AREA "close"))

FOOTNOTES

Page 8

[1] An instructor gives the initial example and classifies the subsequent examples in this scenario as positive or negative. This scenario focuses on the learning method rather than interaction.

Page 14

[2] The infantry platoons are slightly to the northwest of hill-sector 868-1 and are shown as a clump of very small circles.

Page 19

[3] The initial positive example is always used as a reference, so that the user can generate a *positive* explanation, which will correct the current procedure. Explanations of why the negative example is incorrect can also be generated, but currently cannot be used to modify the plausible version space. Such *negative* explanations would have to be converted to *positive* explanations. An additional benefit to referring back to the initial example is that the user is quite familiar with it, as the user specified it at the start of the learning process as a prototypical solution.

Page 22

[4] This involves saving the procedure and then loading it again to generate a new search space without any fixed variables.

FIGURE CAPTIONS

Fig. 1. Captain Design.

Fig. 2. PVS Procedure.

Fig. 3. Example of Plausible Version Space Procedures

Fig. 4 Terrain Map

Fig. 5: Constructing Captain Using the Agent-Disciple Toolkit.

Fig. 6. ModSAF Defensive Mission Scenario.

Fig. 7. Initial Placement on ModSAF Map by User.

Fig. 8. Initial Placement Problem / Solution.

Fig. 9. Initial Procedure Formed by Captain.

Fig. 10. Negative Company D Placement Generated by Captain.

Fig. 11. Negative Company D Problem / Solution.

Fig. 12. Positive Company D Placement Generated by Captain.

Fig. 13. Positive Company D Problem / Solution

Fig. 14. Final Procedure Formed by Captain.

Fig. 15. Placement of Platoons by a Captain Agent for a Defensive Mission.

Fig. 16. Reduction of Search Space by Explanations

Correspondences Abstract

The current practice of building an agent involves a developer programming it for each task it must perform, but agents constructed in this manner are difficult to modify and cannot be trained by a user. Agent-Disciple is a system for training *instructable* agents through user-agent interaction. We report here on our work that uses Agent-Disciple to provide a learning agent that can command simulated military forces, which currently have many limitations in modeling human behavior. We present an instructable Company Commander Agent that can be trained interactively to perform various military missions using the Captain system based on Agent-Disciple.