Teaching Virtual Experts for Multi-Domain Collaborative Planning

Gheorghe Tecuci, Mihai Boicu, Dorin Marcu, Marcel Barbulescu, Cristina Boicu, Vu Le, Thomas Hajduk
Learning Agents Center, George Mason University, 4400 University Drive, Fairfax, VA 22030, USA.
{tecuci, mboicu, dmarcu, mbarbule, ccascava, vle3}@gmu.edu, THajduk@aol.com, http://lac.gmu.edu

Abstract— **This paper presents an approach to rapid development of virtual planning experts that can collaborate to develop plans of action requiring expertise from multiple domains. The approach is implemented into a new type of software tool, called Disciple-VPT, which includes an extensible library of virtual planning experts from different domains. Teams of such virtual experts can be rapidly assembled from the library to generate complex plans of actions that require their joint expertise. The basic component of the Disciple-VPT tool is the Disciple-VE learning agent shell that can be taught directly by a subject matter expert how to plan, trough planning examples and explanations, in a way that is similar to how the expert would teach an apprentice. Copies of the Disciple-VE shells are used by experts in different domains to rapidly populate the library of virtual experts of Disciple-VPT.**

Index Terms—**knowledge engineering, HTN planning, learning, collaboration, expert systems, emergency response planning, knowledge bases, ontology, rules, software tool**

I. Introduction

This paper presents an approach to rapid development of virtual experts for application areas requiring multi-domain collaborative planning. A ***virtual expert*** *is a knowledge-based agent that can rapidly acquire planning expertise from a subject matter expert and can collaborate with other virtual experts to develop plans that are beyond the capabilities of individual virtual experts*.

In this paper, ***planning*** *means finding a partially ordered set of elementary actions that perform a complex task* [1].

The multi-domain collaborative planning approach is being implemented into a new type of software tool, called Disciple-VPT (Virtual Planning Team), which consists of a library of virtual experts and a generic software shell for creating multi-agent planning systems for a variety of application domains. A representative application of Disciple-VPT is planning the response to emergency situations, such as, a tanker truck leaking toxic substance near a residential area, a propane truck explosion, a bio hazard, an aircraft crash, a natural disaster or a terrorist attack [2]. The US National Response Plan [3] identifies 15 primary emergency support functions performed by federal agencies in emergency situations. Similarly local and state agencies undertake these functions responding to such emergencies without or before any federal assistance is provided. Each such function defines an expertise domain, such as emergency management; police operations; fire department operations; hazardous materials handling; health and emergency medical services; sheltering, public works and facilities; and federal law enforcement. In this case, the library of Disciple-VPT will include virtual experts corresponding to these domains.

Critical to the generality and usefulness of Disciple-VPT is its ability to rapidly develop virtual experts that have the expertise of specific human experts from a wide variety of domains. Disciple-VPT incorporates the Disciple-VE system, a learning agent shell that can be taught directly by a subject matter expert how to plan, for instance, by showing it how to plan the performance of a specific task and helping it understand the reasoning process. As a result, the agent learns general planning rules from such planning examples and builds its knowledge base. In time, the expert-agent interaction evolves from a teacher-student interaction toward an interaction where both collaborate in planning. During this joint planning process, the agent learns not only from the contributions of the expert, but also from its own successful or unsuccessful planning attempts.

Disciple-VE builds on the previous versions of the Disciple learning agent shell that were used to develop agents for course of action critiquing and center of gravity analysis, and were successfully evaluated as part of DARPA's High Performance Knowledge Bases and Rapid Knowledge Formation programs [4]. The Disciple agents for center of gravity analysis have been used in several courses at the US Army War College [5] and the Air War College. The Disciple approach has been significantly extended to develop the Disciple-VPT software tool, as presented in the rest of this paper.

The next section presents the general architecture of Disciple-VPT and discusses the different possible uses of this general and flexible tool. Sec. III describes a sample scenario from the emergency response planning area, which is used to present the features of Disciple-VPT. Sec. IV presents the architecture of the Disciple-VE learning agent shell which is at the basis of the capabilities of Disciple-VPT. Sec. V presents the learning-oriented knowledge representation of Disciple-VE. Sec. VI presents the hierarchical task-network (HTN) planning performed by the Disciple virtual experts. After that, Sec. VII presents a modeling language and methodology developed to help a subject matter expert explain to a Disciple agent how to plan, by using the task reduction paradigm. Sec. VIII discusses how a Disciple-VE agent can perform complex inferences as part of a planning process.

The next two sections, IX and X, present the teaching and learning methods of Disciple-VE, first for inference tasks and then for planning tasks. Sec. XI presents the organization of the library of virtual experts of Disciple-VPT. After that, Sec. XII presents Disciple-VPT's approach to multi-agent collaboration. Sec. XIII discusses the development of two virtual experts, one for fire operations and the other for emergency management. Sec. XIV presents the evaluation results, and Sec. XV summarizes our research contributions and the future research directions.

II. The Architecture of Disciple-VPT

Fig. 1 presents the end-user's view of the three major components of Disciple-VPT:

- ***VE Assistant***, an agent that supports the user in using Disciple-VPT;
- ***VE Library***, an extensible library of virtual planning experts;
- ***VE Team***, a dynamically assembled team of virtual experts selected from the VE Library.

The user interacts with the VE Assistant to specify a situation and the profiles of several human experts that may collaborate to plan the achievement of various goals in that situation. Next, a team of virtual planning experts with similar profiles is automatically assembled from the VE Library. This VE Team then simulates the planning performed by the human experts, generating plans for achieving various goals in the given situation.

Disciple-VPT allows the development of collaborative planners for a variety of applications by populating its library with corresponding virtual experts. For instance, planning the response to emergency situations requires virtual experts for emergency management, hazardous materials handling, federal law enforcement, etc. Other application areas, such as planning of military operations, require a different set of virtual experts in the VE Library. Moreover, for a given type of task and application area, different multi-domain planning systems can be created by assembling different teams of virtual experts.

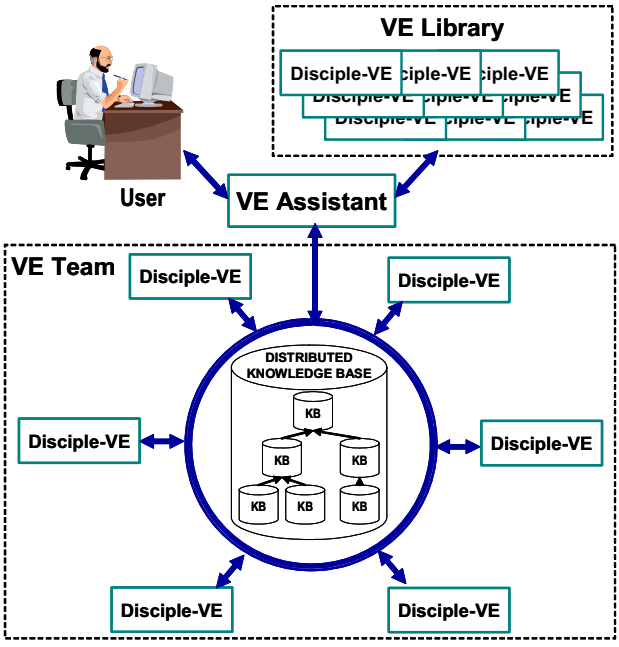

Figure 1. Overall architecture of Disciple-VPT.

There are many ways in which a fully-functional Disciple-VPT system can be used for training or actual planning assistance. For instance, in the context of emergency response planning, it can be used to develop a wide range of training scenarios by guiding the user to select between different scenario characteristics. Disciple-VPT can also be used to assemble teams of virtual planning experts that can demonstrate and teach how people should plan the response to various emergency situations. Another approach is to assemble combined teams which include both people and virtual experts. The team members will then collaborate in planning the response to the generated emergency scenario. In a combined team, human responders can play certain emergency support functions by themselves, or can play these functions with the assistance of corresponding virtual experts. During the training exercise a responder who has a certain emergency support function will learn how to perform that function from a corresponding virtual expert with higher competence. The responder will also learn how to collaborate with the other responders or virtual experts that perform complementary support functions.

The Disciple-VPT approach to expert problem solving extends significantly the applicability of the traditional expert systems [6-10]. Such an expert system is limited to a narrow expertise domain and its performance decreases dramatically when attempting to solve problems that have elements outside its domain of expertise. On the contrary, a Disciple-VPT type system can efficiently solve such problems by incorporating additional virtual experts. Because many expert tasks actually require collaboration with other experts, a Disciple-VPT type system is more suitable for solving real-world problems.

The next section introduces in more detail the scenario from the emergency response planning area that guided the development of Disciple-VPT.

III. Emergency Response Planning

Emergency Response Planning was introduced in the previous sections. A sample emergency situation which will be used in the rest of this paper is the following one:

"Workers at the Propane bulk storage facility in Gainsville, Virginia, have been transferring propane from a train car to fill one of two 30,000 gallon bulk storage tanks. A fire is discovered in the fill pipe at the bulk tank and a large fire is developing. The time is 15:12 on a Wednesday in the month of May. The temperature is 72 degrees and there is a light breeze out of the west. The roads are dry and traffic volume is moderate. The fire department is summoned to the scene 5 minutes after the fire started. The facility is located in a rapidly growing area 2,000 ft from an interstate highway and 200 ft from two heavily traveled US highways. New shopping centers have popped up in the area including food stores, large box building supply facilities, and large box retail facilities. As always, these facilities are accompanied by fast food restaurants and smaller retail stores. Residential concentrations include approximately 2400 residents. The

Local Emergency Operations Plan has all the required components including Public Information, Communications, and Sheltering. Shelters utilize schools managed by the Red Cross. The Virginia Department of Transportation provides highway services."

Planning the appropriate response to this emergency situation requires the collaboration of experts in fire department operations, in emergency management, and in police operations. The generated plan will consist of hundreds of partially order actions.

One group of actions deal with the arrival of resources, such as fire units, emergency management services units, police units, as well as individuals with different areas of expertise (e.g. emergency manager, safety officer, highway supervisor, planning officer, training logistics officer, public information officer).

Another group of actions deal with the establishment of the structure of the Incident Command System (ICS) and the allocation of resources based on the evaluation of the situation. The structure of the ICS follows the standard U.S. National Incident Management System [11]. The National Incident Management System establishes standard incident management processes, protocols and procedures so that all local, state, federal and private-sector emergency responders can coordinate their responses, share a common focus and more effectively resolve events. Its main components are the unified command, the command staff, and the general staff. The structure and organization of these components depend on the current situation. For example, in the case of the above scenario, the unified command includes representatives from the fire department, police department, highway department, and propane company. The command staff includes a safety officer, a public information officer and a liaison officer. The general staff includes an operation section, a planning section, a logistics section, and a finance and administration section. Each of these sections is further structured and staffed.

Yet other groups of actions deal with the various activities performed by the components of the Incident Command System. For instance, in the case of the above scenario, the fire management group may perform the cooling of the propane tank with water. The evacuation branch may evacuate the Gainsville hot zone. The emergency manager may arrange for transportation, sheltering, and emergency announcements to support the evacuation. The Gainsville perimeter control branch implements the perimeter control for the Gainsville hot zone. The Gainsville traffic control branch implements the traffic control to facilitate the evacuation of the Gainsville hot zone. The Gainsville command establishes rapid intervention task forces to respond if the propane tank explodes.

One difficulty in generating such a plan, apart from the fact that it involves many actions, is that the actions from the above groups are actually performed in parallel. The goal of the research presented in this paper is to create a capability for rapid and low cost development of virtual planning experts to be used in this type of multi-domain collaborative planning. Moreover, the plans generated by the system should be more comprehensive than those produced by a collaborative team of humans, and should be generated much faster and cheaper than currently possible. The next section introduces the Disciple-VE learning agent shell which is at the basis of Disciple-VPT.

IV. The Disciple-VE Learning Agent Shell

Disciple *denotes an evolving theory, methodology and family of tools for the development of knowledge-based agents by subject matter experts, with limited assistance from knowledge engineers* [12-14]. The main goal of the Disciple approach is to overcome the knowledge acquisition bottleneck in the development of knowledge-based systems [6]. Its basic idea is to develop a general problem solving and learning agent that has no specific knowledge in its knowledge base, but can be taught directly by a subject matter expert, and can develop its knowledge base to become an expert system. We call such an agent, a ***learning agent shell.*** The Disciple-VE learning agent shell extends significantly the previous versions by incorporating capabilities of learning for planning and by allowing rapid development of knowledge-based planners, as will be discussed in more details in the rest of this paper.

The general problem solving paradigm of a Disciple agent is task reduction [12, 15-17]. *In the* ***task reduction paradigm,*** *a complex problem solving task is successively reduced to simpler tasks, solutions of the simplest tasks are found, and these solutions are successively combined into the solutions of the initial task,* as illustrated in Fig. 7. In the context of planning, this approach reduces to Hierarchical Task Network (HTN) planning where the initial complex task is reduced to a partially ordered set of elementary actions [1, 18-20]. This process is illustrated in Fig. 5. In the case of Disciple-VE, planning tasks are integrated with inference tasks, which significantly increases the power of HTN planning.

In order to perform HTN planning and inference, the knowledge base of Disciple-VE contains two main types of knowledge: an object ontology and a set of reasoning rules. The object ontology, which is described in more details in the next section, represents the types of objects from an application domain, together with their properties and relationships [21, 22]. A fragment of the object ontology for emergency planning is shown in Fig. 2.

The reasoning rules are expressed with the elements of the object ontology. ***Reduction rules*** *indicate how general planning or inference tasks can be reduced to simpler tasks, actions, or solutions.* ***Synthesis rules*** *indicate how solutions of simpler tasks can be combined into solutions of complex tasks, or how actions can be combined into partially ordered plans for more complex tasks.* The main strength of the Disciple agents comes from the fact that they can easily and rapidly learn rules (such as those in Fig. 15) from subject matter experts.

The Disciple-VE shell is used to rapidly develop a Disciple-VE agent for a specific planning domain by following a two phase process:

- The development of an initial object ontology for that domain, which is performed jointly by a knowledge engineer and a subject matter expert.

- The teaching of Disciple-VE, which is performed by the subject matter expert, with limited assistance from the knowledge engineer.

The subject matter expert teaches Disciple-VE how to plan the performance of a complex task in a way that is similar to how the expert would teach a person. For instance, the expert will show and explain the agent how to plan the performance of "Respond to Gainsville incident" and the agent will learns general planning and inference rules for this type of task. This process is based on:

- ***Mixed-initiative planning*** [23, 24], where the expert develops the more creative parts of the plan and the agent develops the more routine ones.
- ***Integrated learning and teaching*** [13], where the expert helps the agent to learn (e.g. by providing examples, hints and explanations), and the agent helps the expert to teach it (e.g. by asking relevant questions).
- ***Multistrategy learning*** [25], where the agent integrates complementary strategies, such as learning from examples, learning from explanations, and learning by analogy, to learn general concepts and rules.

The integrated learning, planning, and inference capabilities of Disciple-VE are based on a learning-oriented knowledge representation which is presented next.

V. Learning-Oriented Knowledge Representation

A. The Object Ontology

At the basis of Disciple's learnable knowledge representations are the notions of instances, concepts and generalization. *An **instance** is a representation of a particular entity in the application domain* (e.g. Gainsville incident, from the bottom-left of Fig. 2). *A **concept** is a representation of a set of instances.* For example, the major fire emergency concept represents all the incidents that are major fire emergencies. One such instance is Gainsville incident. As shown in Fig. 2, this information is represented as "Gainville incident instance_of major fire emergency".

*A concept P is said to be **more general than** (or a generalization of) another concept Q if and only if the set of instances represented by P includes the set of instances represented by Q.* For example, major emergency is more general than major fire emergency, which is expressed as "major fire emergency subconcept_of major emergency."

The instances and concepts are organized into generalization hierarchies like the one from Fig. 2. These structures are not strict hierarchies, meaning that a concept may be a subconcept of several concepts (e.g. propane is both a chemical substance and a hazardous substance).

The instances and concepts may have features representing their properties and relationships. For example, "Gainsville incident is_caused_by fire1" and "fire1 is_fuelled_by gas propane f1", as illustrated in Fig. 2. The bottom part of Fig. 3 shows all the features of fill pipe1 in the interface of the Association Browser of Disciple.

Each feature, such as is_fuelled_by, is characterized by a domain and a range. *The **domain** of a feature is a concept that represents all objects that may have that feature. The **range** of a feature is a concept that represents all the possible values of that feature.* For example, the domain of is_fuelled_by is fire and its range is hazardous substance.

The features are also organized into a generalization hierarchy. For example, the top part of Fig. 3 shows (a rotated view of) a fragment of the feature hierarchy, in the interface of the Hierarchy Browser of Disciple. In this hierarchy the feature has_as_part (shown in the left hand side of Fig. 3) is more general than has_as_member which, in turn, is more general than has_as_supervisor.

Together, the object hierarchy and the feature hierarchy represent the object ontology of a Disciple-VE agent. Thus, *the **object ontology** is a hierarchical representation of the objects from the application domain, representing the different kinds of objects, the properties of each object, and the relationships existing between objects.*

In general the object ontology does not contain all the relevant concepts and instances from the application domain and is therefore incomplete. Also the representation of a given concept or instance may by not include all its relevant features, being itself incomplete. Such an object ontology will have to be extended by the agent during the planning and learning process.

The object ontology plays a crucial role in Disciple, being at the basis of knowledge representation, user-agent communication, planning, knowledge acquisition, and learning, as discussed in the following.

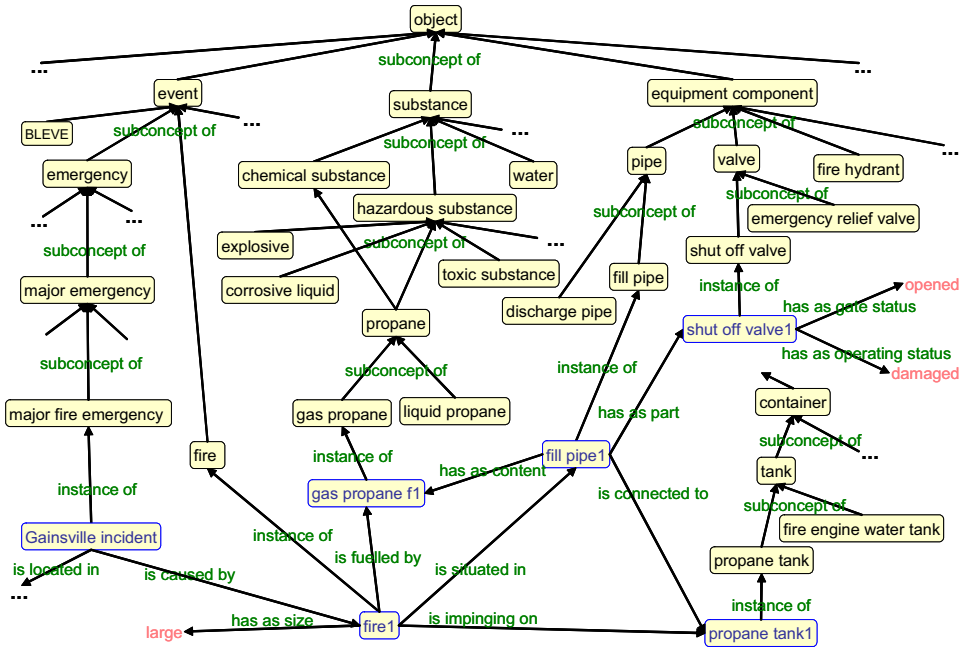

Figure 2. Fragment of the object ontology from the emergency planning area.

B. Concept Representation

Using the (object and feature) concepts from the object ontology, one can define more complex concepts as logical expressions involving these primary concepts. The basic representation unit (BRU) for a more complex concept has the form $\{?O_1, ?O_2, \ldots, ?O_n\}$, where each $?O_i$ has the structure indicated in (1).

$$\begin{array}{lll} ?O_i & \text{is} & \text{concept}_i \\ & \text{feature}_{i1} & ?O_{i1} \\ & \ldots & \\ & \text{feature}_{im} & ?O_{im} \end{array} \quad (1)$$

Concept_i is an object concept from the object ontology, a numeric interval, or a list of strings, and $?O_{i1} \ldots ?O_{im}$ are distinct variables from the set $\{?O_1, ?O_2, \ldots, ?O_n\}$. For example, the concept "fire fuelled by gas propane" is represented by the pair $\{?O_1, ?O_2\}$, where $?O_1$ is a fire fuelled by $?O_2$, and $?O_2$ is gas propane, as indicated by the expression (2).

$$\begin{array}{lll} ?O_1 & \text{is} & \text{fire} \\ & \text{is\_fuelled\_by} & ?O_2 \\ ?O_2 & \text{is} & \text{gas propane} \end{array} \quad (2)$$

In general, a concept may be a conjunctive expression of form (3), meaning that its instances satisfy BRU and do not satisfy BRU_1 and … and do not satisfy BRU_p.

$$\text{BRU \& not BRU}_1 \text{ \& ... \& not BRU}_p \quad (3)$$

However, instead of "not" we write "Except When". For example, expression (4) represents the concept "fire fuelled by gas propane where the fire is not small".

$$\begin{array}{lll} ?O_1 & \text{is} & \text{fire} \\ & \text{is\_fuelled\_by} & ?O_2 \\ ?O_2 & \text{is} & \text{gas propane} \\ \textit{Except When} & & \\ ?O_1 & \text{is} & \text{fire} \\ & \text{has\_as\_size} & \text{small} \end{array} \quad (4)$$

C. Generalization and Specialization Rules

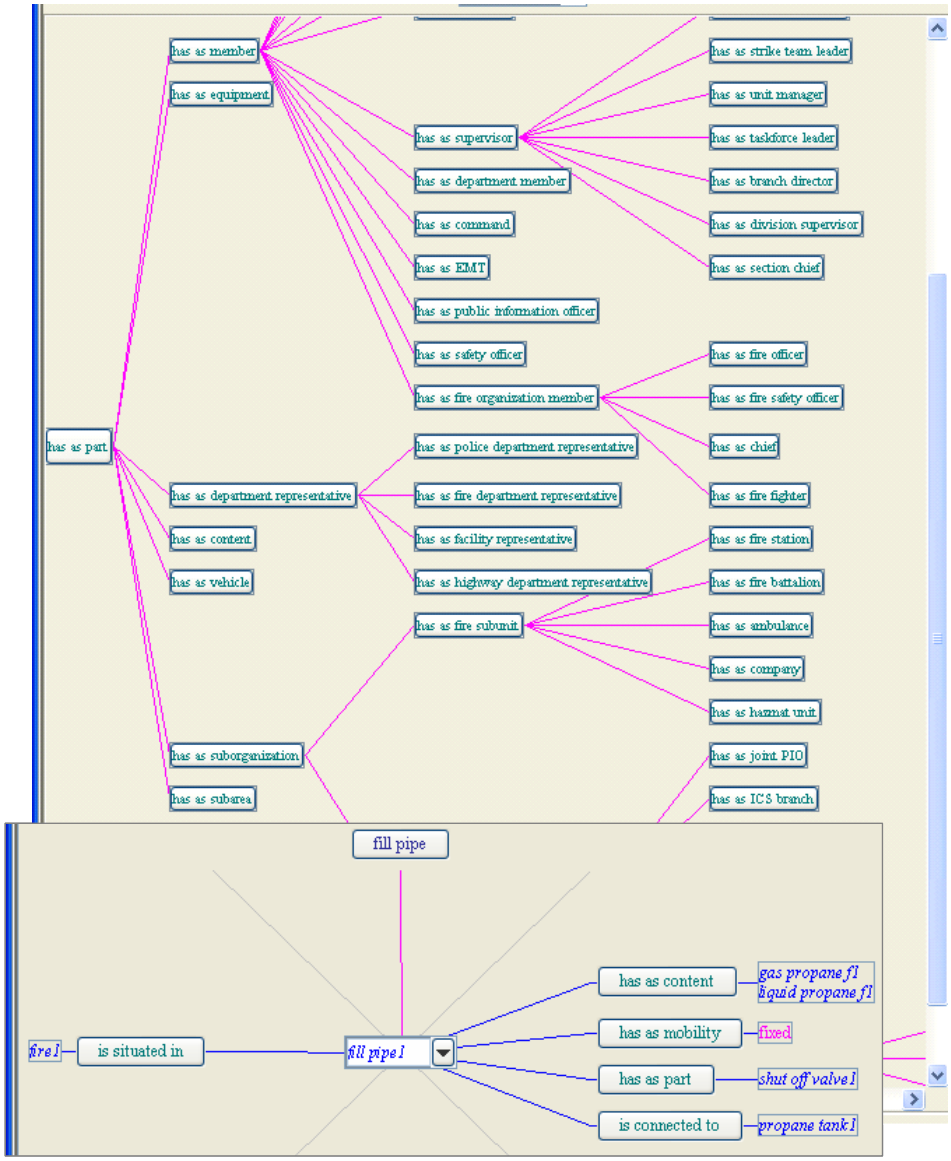

Figure 3. Feature hierarchy (top) and instance description (bottom).

The object ontology is at the basis of the generalization language for learning, as discussed in the following. A concept, such as (2), may be generalized or specialized by using generalization or specialization rules.

*A **generalization rule** is a rule that transforms a concept into a more general concept.* The reverse of any generalization rule is *a **specialization rule** which transforms a concept into a less general concept.*

Examples of generalization rules are climbing the generalization hierarchy, dropping conditions, turning numbers into intervals, and generalizing to feature domains and ranges [13].

*The **climbing generalization hierarchy rule** generalizes a concept by replacing a concept from its description with a more general concept.* For example, by replacing gas propane with propane in (2), the concept "fire is_fuelled_by gas propane" is generalized to "fire is_fuelled_by propane".

*The **dropping condition rule** generalizes a concept by eliminating a constraint from its description.* For example, by eliminating the Except When condition

from (4), the concept "fire is_fuelled_by gas propane where the fire is not small" is generalized to "fire is_fuelled_by gas propane".

Let us consider the fact "fire1 is_fuelled_by gas propane f1" which is represented at the bottom of Fig. 2. This fact can be rewritten as shown in (5).

$?O_1$	is	fire1	(5)
	is_fuelled_by	$?O_2$	
$?O_2$	is	gas propane f1	

As indicated above, the domain and the range of the feature is_fuelled_by are fire and hazardous substance, respectively. Therefore, a maximally general generalization of "fire1 is_fuelled_by gas propane f1" is "fire is_fuelled_by hazardous substance." This is obtained by applying the ***generalizing to feature domains and ranges rule***, *where the entity which has a feature f* (e.g. fire1) *is generalized to the domain of the feature* (e.g. fire*), and the entity which is the value of the feature f* (e.g. gas propane f1) *is generalized to the range of that feature.* This rule is important because it imposes limits on how much an expression can be generalized.

During learning, Disciple-VE learns general concepts and rules by applying such generalizations and specialization rules, as discussed in the following sections.

D. Plausible Version Spaces

Disciple-VE learns general reasoning rules (or concepts) staring from a single example of a reasoning step. *During the learning process, Disciple-VE maintains a set of possible versions of the rule (concept) to be learned, called a* ***version space*** [13, 26]. The concepts in this space are partially ordered, based on the ***"more general than"*** relationship defined in Sec. V.A. A concept from this space can be obtained from another concept from the space by applying generalization or specialization rules. For that reason, the version space can be represented by an upper bound and a lower bound. The ***upper bound*** *of the version space contains the most general concepts from the version space and the* ***lower bound*** *contains the least general concepts.* Any concept which is more general than a concept from the lower bound and less general than a concept from the upper bound is part of the version space and may be the actual concept to be learned. A *version space may be regarded as a* ***partially learned concept***.

The version spaces built by Disciple-VE during the learning process are called plausible version spaces because their upper and lower bounds are generalizations based on an incomplete object ontology. Therefore a plausible version space is only a plausible approximation of the concept E_h to be learned, as illustrated in Fig. 4.

The plausible upper bound of the version space from the right hand side of Fig. 4 contains two concepts: "fire fuelled by hazardous substance" and "fire fuelled by chemical substance", as shown by (6).

$?O_1$	is	fire	(6)
	is_fuelled_by	$?O_2$	
$?O_2$	is	hazardous substance	
and			
$?O_1$	is	fire	
	is_fuelled_by	$?O_2$	
$?O_2$	is	chemical substance	

Similarly, the plausible lower bound of this version space contains two concepts, "fire fuelled by gas propane" and "fire fuelled by liquid propane".

The concept E_h to be learned (shown in the left hand side of Fig. 4) *is,* ***as an approximation****, less general than* ***one*** *of the concepts from the plausible upper bound. Also, E_h is,* ***as an approximation****, more general than* ***any*** *of the concepts from the plausible lower bound.* As Disciple-VE encounters additional positive and negative examples of the concept (rule) to be learned, it generalizes and/or specializes the two bounds so that they converge toward one another and approximate E_h better and better. This behavior is different from that of the version spaces introduced by [26], where ***one*** of the concepts from the upper bound ***is always more general*** than the concept to be learned (and the upper bound is always specialized during learning), and ***any*** of the concepts from the lower bound ***is always less general*** than the concept to be learned (and the lower bound is always generalized during learning). The major difference is that the version spaces introduced by [26] are based on a complete representation space that includes the concept to be learned. On the contrary, the representation space for Disciple is based on an incomplete and evolving object ontology, as mentioned above. Therefore, Disciple addresses the more complex and more realistic problem of learning in the context of an evolving representation space.

The notion of plausible version space is fundamental to the knowledge representation, problem solving, and learning methods of Disciple because all the partially learned concepts are represented using this construct. For instance, a partially learned feature has its domain and range represented as plausible version spaces. Similarly, as discussed in more details in Sec. IX and X, a partially learned rule is also represented as a plausible version space.

The next section introduces the type of hierarchical task-network planning performed by Disciple-VE and the associated elements that are represented into its knowledge base.

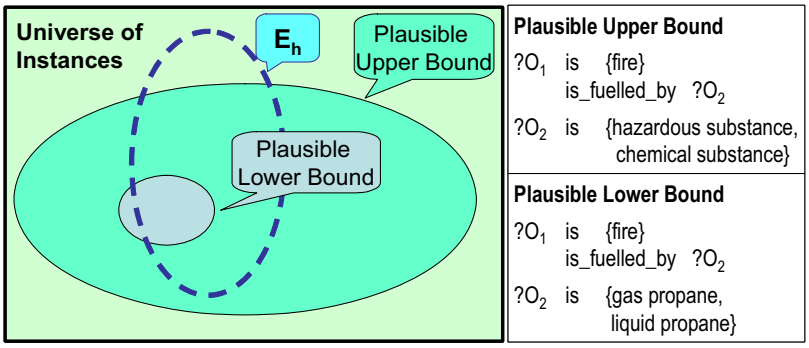

Figure 4. A plausible version space for a partially learned concept.

VI. HIERARCHICAL TASK NETWORK PLANNING

The planning paradigm used by Disciple-VE is HTN planning [1], extended to facilitate agent teaching and learning, and mixed-initiative planning.

*The goal of an **HTN planner** is to find a partially order set of elementary actions that perform a complex task, by successively decomposing the task into simpler and simpler tasks, down to the level of elementary actions.* HTN planning is the planning approach that has been used for practical applications more than any other approach because it is closer to how human experts think about when solving a planning problem.

We will illustrate the HTN planning process performed by a Disciple-VE agent with the abstract example from Fig. 5. In this example, Planning Task 1 is reduced to Planning Task 2 and Planning Task 3. This means that by performing Planning Task 2 and Planning Task 3 one accomplishes the performance of Planning Task 1. Because Planning Task 2 is reduced to Action 1 and Action 2, and Planning Task 3 is reduced to Action 3, a plan for performing Planning Task 1 consists of Action 1, Action 2, and Action 3.

There are two types of reductions, task decomposition and task specialization. ***Task decomposition*** *means breaking a task into a partially ordered set of subtasks and/or actions.* ***Task specialization*** *means reducing a task to a more detailed task or to an action.*

The tasks or actions in a decomposition can be partially ordered. For instance, in Fig. 5, Action 2 has to be performed after Action 1 has been performed. Notice also that there is no order relation between Planning Task 2 and Planning Task 3. This means that these tasks may be performed in parallel or in any order. Stating that Planning Task 3 is performed after Planning Task 2 would mean that any subtask or action of Planning Task 3 has to be performed after any subtask or action of Planning Task 2. Formulating such order relations between the tasks significantly increases the efficiency of the planning process because it reduces the number of partial orders that it has to consider. On the other hand, it also reduces the number of generated plans, if the tasks should not to be ordered.

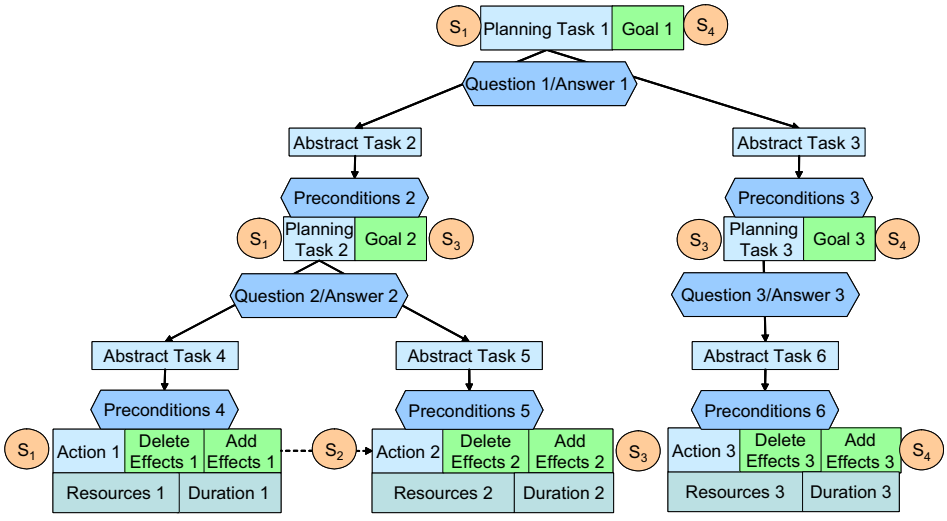

Figure 5. Hierarchical task network planning example.

Planning takes place in a given world state. *A **world state** is represented by all the objects present in the world together with their properties and relationships at a given moment of time.* For instance, the bottom part of Fig. 2 shows a partial representation of a world state where fire1, which is situated in fill pipe1, is impinging on propane tank1. As will be discussed in more details in Sec. XI, each world state is represented by Disciple-VE as a temporary state knowledge base.

The states are changed by the performance of elementary actions. Abstract representations of actions are shown at the bottom of Fig. 5. *An **action** is characterized by name, preconditions, delete effects, add effects, resources and duration. An action can be performed in a given world state S_i if the action's preconditions are satisfied in that state. Action's execution has a duration and requires the use of some resources. The resources are objects from the state S_i that are uniquely used by this action during its execution. This means that any other action that would need some of these resources cannot be executed in parallel with it. As a result of action's execution the state S_i changes into the state S_j, as specified by the action's effects. The delete effects indicate what facts from the initial state S_i are no longer true in the final state S_j. The add effects indicate what new facts become true in the final state S_j.*

An action from the emergency planning area is shown in the bottom-right pane of Fig. 6. The action's preconditions, name, delete and add effects are represented as natural language phrases that contain instances, concepts and constants from the agent's ontology. Action's duration can be a constant, as in this example, or a function of the other instances from the action's description. Resources are represented as a list of instances from the ontology. The starting time is computed by the planner.

*A **goal** is a representation of a partial world state.* It specifies what facts should be true in a world state so that the goal is achieved. As such, a goal may be achieved in several world states.

*A **task** is characterized by name, preconditions and goal. A task is considered for execution in a given world state if its preconditions are satisfied in that state. Successful execution of the task leads to a new world state in which the task's goal is achieved.* Unlike actions, tasks are not executed directly, but are first reduced to actions which are executed.

The top-right part of Fig. 6 shows a task reduction tree in the interface of the Reasoning Hierarchy Browser of Disciple. The initial task "Respond to the Gainsville incident" is reduced to five subtasks. The second of these subtasks (which is outlined in the

figure) is successively reduced to simpler subtasks and actions. The left part of Fig. 6 shows the reduction of the initial task in the interface of the Reasoning Step Editor, which displays more details about each task. As in the case of an action, the task's name, preconditions and goal are represented as natural language phrases that include instances and concepts from the agent's ontology. Notice that none of the visible "After" boxes is checked, which means that Sub-task(1), Sub-task(2), and Sub-task(3) are not ordered.

The single most difficult agent training activity for the subject matter expert is to make explicit how s/he solves problems, by using the task reduction paradigm, activity which we call ***modeling expert's reasoning****.* To cope with this problem, we have developed an intuitive modeling language, a set of modeling guidelines, and a set of modeling modules which help the subject matter experts to express his/her reasoning [27]. However, planning introduces additional complexities related to reasoning with different world states and with new types of knowledge elements, such as preconditions, effects, and goals. For these reasons, and to facilitate agent teaching by a subject matter expert, we have extended both the modeling approach of Disciple and the classical HTN planning paradigm [1], as discussed in the next section.

VII. A Modeling Language for HTN Planning

To teach the agent how to plan the expert has to first show the agent an example in the form of a planning tree like the ones in Fig. 5 and Fig. 6. The expert formulates the initial task (e.g. Planning Task 1 in Fig. 5) and then follows a systematic procedure to develop a detailed plan of actions that perform the initial task. S/He follows a task reduction paradigm where the initial task is successively reduced to simpler and simpler tasks, down to the level of elementary actions. The partially ordered set of these elementary actions represents the plan for performing the initial task.

As illustrated in Fig.5 and Fig. 6, the task reduction process is guided by questions and answers, as if the expert is asking himself/herself how to reduce the current task. Consider, for instance, a task that may be reduced (i.e. performed) in different ways. Then the question should be related to the factors that determine the reduction strategy to choose. Therefore the answer will help the expert to choose the strategy and define the reduction. If there is only one way to reduce the current task, then no Question/Answer pair is necessary. The above strategy is summarized by the following modeling guideline.

Guideline 1: *When reducing a task, ask a question related to the reduction strategy to use. Find the answer to the question and then reduce the task appropriately.*

Notice that Planning Task 1 from Fig. 5 has to be performed in the initial state S_1. Planning Task 2 is also performed in the state S_1 and its preconditions have to be satisfied in that state. Similarly, Action 1 has to be performed in the state S_1. However, this action changes the world state from S_1 to S_2. Therefore Action 2 has to be performed in the state S_2 and its preconditions have to be satisfied in that state. Moreover, it also changes the world state to S_3.

What is the state in which Planning Task 3 is executed? Because there is no order relationship between Planning Task 2 and Planning Task 3, Planning Task 3 can, in principle, be executed either in the state S_1, or S_2, or S_3. In reality, some order relationships may be determined by the resources used by the elementary actions. For instance, if both Action 2 and Action 3 need the same resource, they

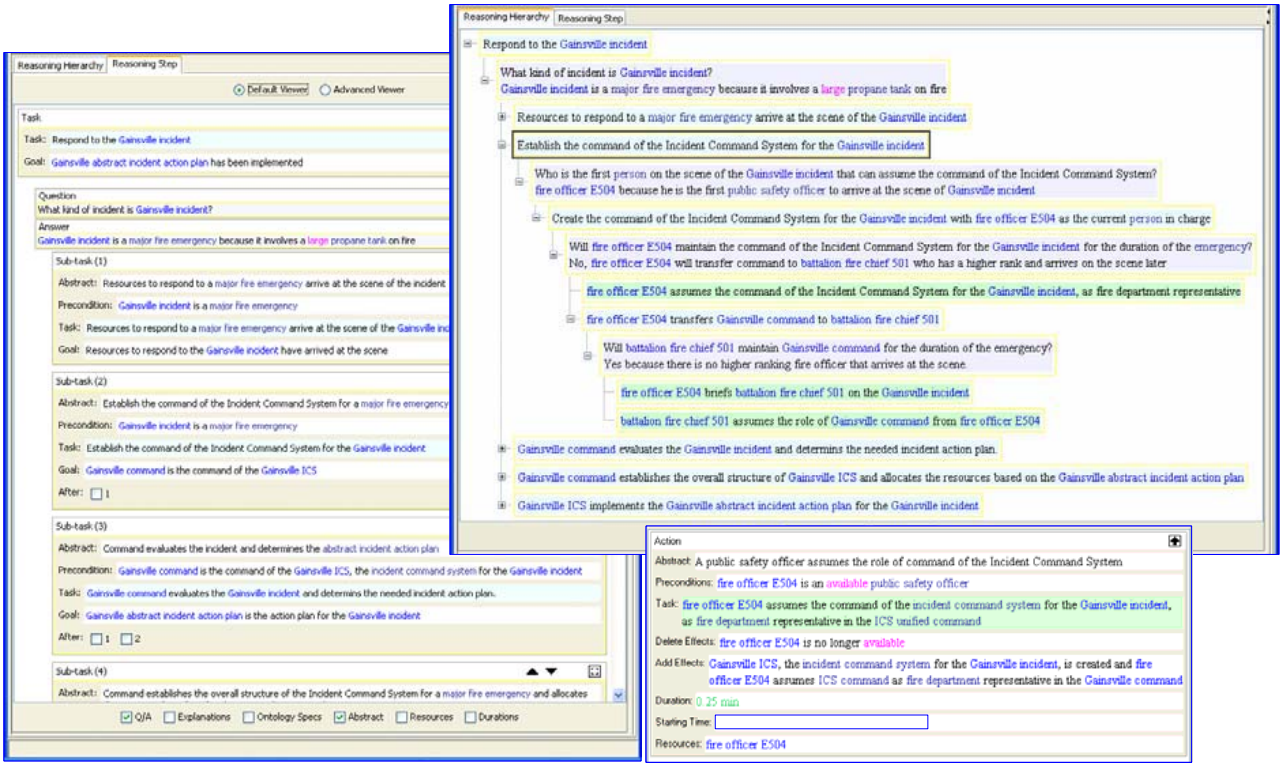

Figure 6. Reasoning hierarchy (top right), reasoning step (left) and elementary action (bottom right).

cannot be executed in parallel.

When showing a planning example to the agent, the expert does not need to consider all the possible ordering relations discussed above, which would be very difficult in the case of the complex problems addressed by Disciple-VE. Instead, the expert has to consider one possible order, such as the one in which Planning Task 3 is executed after Action 2, in state S_3. This allows both the expert and the agent to have a precise understanding of the state in which each task is performed, which is necessary in order to check that its preconditions are satisfied. This leads to the following modeling guideline.

Guideline 2: *When specifying a decomposition of a task into subtasks and/or actions, the expert has to describe the subtasks and actions in a plausible order, even though they can also be performed in a different order.*

The following guideline is intended to facilitate the specification of an entire planning tree by the expert.

Guideline 3: *The expert has to specify the planning tree in a top-down and left-right fashion, indicating a reduction of a task into its subtasks, and continuing with the further reduction of the left-most subtask.*

However, allowing the process specified by the above guideline required the extension of the HTN planning paradigm, as discussed in the following. Let us consider the decomposition of Planning Task 1 from Fig. 5 into Planning Task 2 and Planning Task 3, by following the top-down and left-right process. At the time the expert has to specify this decomposition, s/he knows that Planning Task 2 has to be performed in the state S_1, by s/he does not know which is the state in which Planning Task 3 has to be performed. This state can only be determined after the entire subplan for Task 2 has been specified. In other words, Task 3 can only be specified after the entire subplan for Task 2 has been specified. In order to resolve this contradiction, we have introduced the notion of abstract task.

*An **abstract task** is a simplified specification of a task that does not depend on the actual state in which the task is going to be executed.* As such, an abstract task does not have any precondition, and does not refer to any specific objects or their properties.

An abstract task can be reduced to a concrete task if certain preconditions are satisfied. In principle, the same abstract task may be reduced to different concrete tasks. Therefore the abstract task is not a characteristic of a given concrete task. The following is a guideline for reducing an abstract task to a concrete task.

Guideline 4: *When reducing an abstract task to a concrete task formulate the preconditions to identify those instances and constants from the current world state which are referred in the name of the concrete task, but are not referred in the previous elements of the task reduction step that includes this concretion.*

To illustrate the above guideline, let us consider again the reduction step from the left side of Fig. 6 and the pane labeled Sub-task(3). Notice that the concrete task includes the following instances: Gainsville command and Gainsville incident. Each of these instances appear in the elements listed under Sub-task(2). For example Gainsville command appears in the Goal part. Therefore, according to Guideline 4, no preconditions are required to make the concretion from the abstract task to the concrete task shown in the pane labeled Sub-task(3). However, the expert may still wish (and is allowed) to specify preconditions that identify the instances that appear in the concrete task, as was actually done in this example.

With the introduction of the abstract tasks the expert can now reduce Planning Task 1 to Abstract Task 2 and Abstract Task 3. Then s/he can continue with the reduction of Abstract Task 2 to Planning Task 2, reduction performed in state S_1. Preconditions 2 represents the facts from the state S_1 that are required in order to make this reduction.

After that the expert continues with the reduction of Planning Task 2 to Action 1 and Action 2. Thus Planning Task 2 is actually performed by executing Action 1 and Action 2, which change the world state from S_1 to S_3. At this point the expert can specify the goal achieved by Planning Task 2. This goal is an expression that depends on the effects of Action 1 and Action 2, but is also unique for Task 2, which is now completely specified. Next the expert can continue with planning for Abstract Task 3 in the state S_3.

*The **goal** of a task represents the result obtained if the task is successfully performed.* The main purpose of the goal is to identify those instances or facts that have been added by its component actions and are needed by its follow-on tasks or actions, as indicated in Guideline 5.

Guideline 5: *Specify the goal of the current task such that it includes those instances or facts created by its component actions which are needed in the specification of the follow-on actions or tasks from the current task reduction step.*

To illustrate this guideline, let us consider the Sub-task(2) pane in the left-side of Fig. 6. Notice that the two instances from the Goal part (Gainsville command and Gainsville ICS) are used in the follow-on expressions of the reduction from Fig. 6.

The Reasoning Hierarchy Browser and the Reasoning Step Editor (see Fig. 6) support the modeling process. The Reasoning Hierarchy Browser provides operations to browse the planning tree under development, such as expanding or collapsing it step by step, or in its entirety. It also provides the expert with macro editing operations, such as deleting an entire subtree or copying a subtree and pasting it under a different task. Each reduction step of the planning tree is defined by using the Reasoning Step Editor which includes several editors for specifying the components of a task reduction step. It has completion capabilities that allow easy identification of the names from the object ontology. It also facilitates the viewing of the instances and concepts from the expressions being edited by invoking various ontology viewers.

An important contribution of Disciple-VE is the ability to combine HTN planning with inference, as described in the following section.

VIII. Integration of Planning and Inference

As illustrated in Fig. 5, each planning operation takes place in a given world state and the actions, through their effects, change this state. The planning process is complex and computationally expensive because one has to

keep track of these various world states. However, some operations do not involve the change of the world state, but reasoning about a given state. Let us consider the top level reasoning steps from the right-top pane of Fig. 6 where the task "Respond to the Gainsville incident" is reduced to five subtasks. The third of these subtasks "The Gainsville command evaluates the Gainsville incident and determines the needed incident action plan" is further reduced to two inference actions:

> ***Inference:*** The Gainsville command evaluates the situation created by the Gainsville incident.
> ***Inference:*** The Gainsville command determines the incident action plan for overpressure situation with danger of BLEVE.

The first of these inference actions has as result "overpressure situation with danger of BLEVE in propane tank1 caused by fire1". BLEVE is the acronym of "Boiling Liquid Expanding Vapors Explosion".

From the perspective of the planning process, an ***inference action*** *simulates a complex inference process by representing the result of that process as the add effect of the inference action.* An inference action is automatically reduced to an inference task. *The* ***inference task*** *is performed in a given world state to infer new facts about that state.* These facts are represented as the add effects of the corresponding inference action and added into the world state in which the inference action is performed.

The inference process associated with an inference task

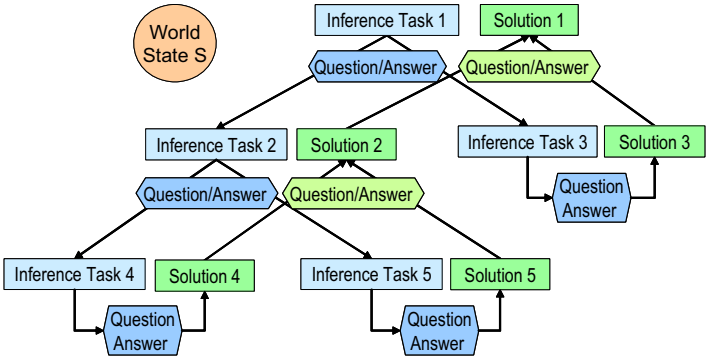

Figure 7. Abstract inference tree.

is also performed by using the task reduction paradigm, but it is much simpler than the planning process because all the reductions take place in the same world state. An abstract example of an inference tree is shown in Fig. 7.

An inference task is performed by successively reducing it to simpler inference tasks, until the tasks are simple enough to find their solutions. Then the solutions of the simplest tasks are successively combined, from bottom-up, until the solution of the initial task is obtained.

This task reduction and solution synthesis process is also guided by questions and answers, similarly to the planning process. Fig. 8 shows the top part of the inference tree corresponding to the task "***Inference:*** Gainsville command determines the incident action plan for overpressure situation with danger of BLEVE." This task is first reduced to two simpler inference tasks: "Determine what can be done to

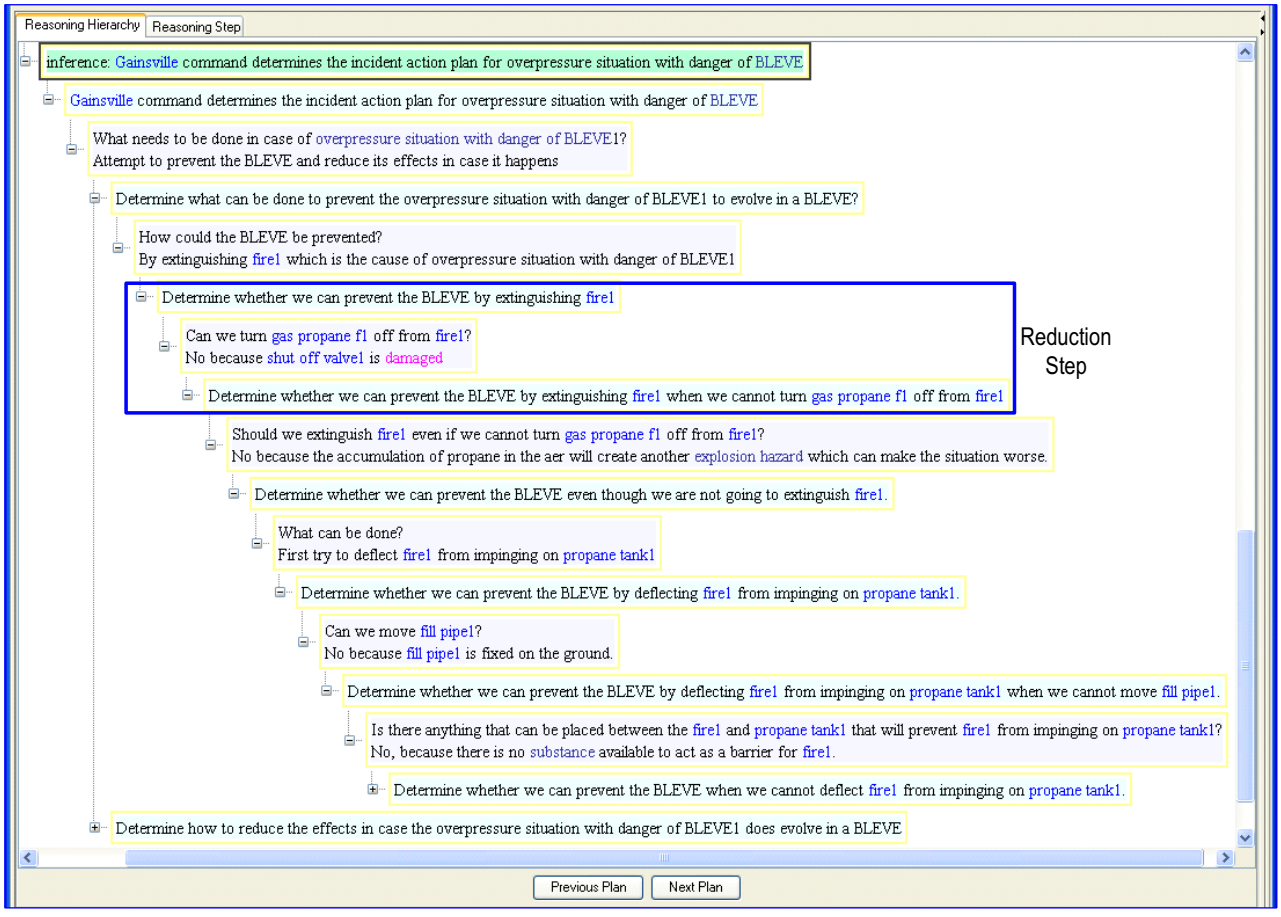

Figure 8. Inference reduction tree.

prevent the overpressure situation with danger of BLEVE to evolve in a BLEVE" and "Determine how to reduce the effects in case the overpressure situation with danger of BLEVE does evolve in a BLEVE." The first of these subtasks is successively reduced to simpler and simpler subtasks, guided by questions and answers, as shown in the top part of Fig. 8.

Notice that an inference tree no longer needs to use elements such as abstract task, preconditions, actions, effects, resources, or duration. The teaching process is also much simpler than in the case of the planning process. Therefore we will first present how the expert can teach Disciple-VE to perform inference tasks. Then we will present how the teaching and learning methods for inference tasks have been extended to allow the expert to also teach the agent how to perform planning tasks.

IX. Teaching an Agent to Perform Inference Tasks

A. Learning Inference Rules

Let us consider again the fragment of the inference tree shown in Fig. 8. During the teaching process the subject matter expert builds this inference tree. Each step in the tree consists of a task, a question, its answer, and one or several subtasks. From each of these steps the agent learns a general task reduction rule. Table I defines the problem of learning these rules.

TABLE I.
THE LEARNING PROBLEM FOR TASK REDUCTION INFERENCE RULES

Given:
- An example E of a reduction step for inference tasks.
- A knowledge base that includes an ontology and a set of rules.
- A subject matter expert who understands why the given example is correct and may answer the agent's questions.

Determine:
- A plausible version space rule R for inference tasks, which is a generalization of E.
- An extended object ontology (if needed for rule learning).

Let us consider the 3rd reduction step from the task reduction tree in Fig. 8, step also shown at the top of Fig. 9. From this task reduction step, Disciple-VE learned the rule shown at the bottom of Fig. 9. This is an IF-THEN rule that preserves the structure and natural language patterns from the example. Indeed, the IF task, the question/answer pair, and the THEN task are generalizations of the corresponding elements from the example where the instances and constants have been replaced with variables. In addition, the rule contains a main condition. An instance of the rule is considered a correct reduction if the corresponding variable values satisfy the main condition.

The rule in Fig. 9 is only partially learned because, instead of a single applicability condition, it contains a plausible version space for it. The plausible lower bound of the applicability condition is the set of the tuples of the rule variable values that are less general than the corresponding elements of the Lower Bound table and satisfy the relationships from the Relationship table. For example, any value of ?O4 should be an instance of shut off valve which has_as_operating_status ?S1 which should have the value damaged. Moreover, this value of ?O4 should be the value of the relationship has_as_part of an instance of ?O5 which should be a fill pipe and should have

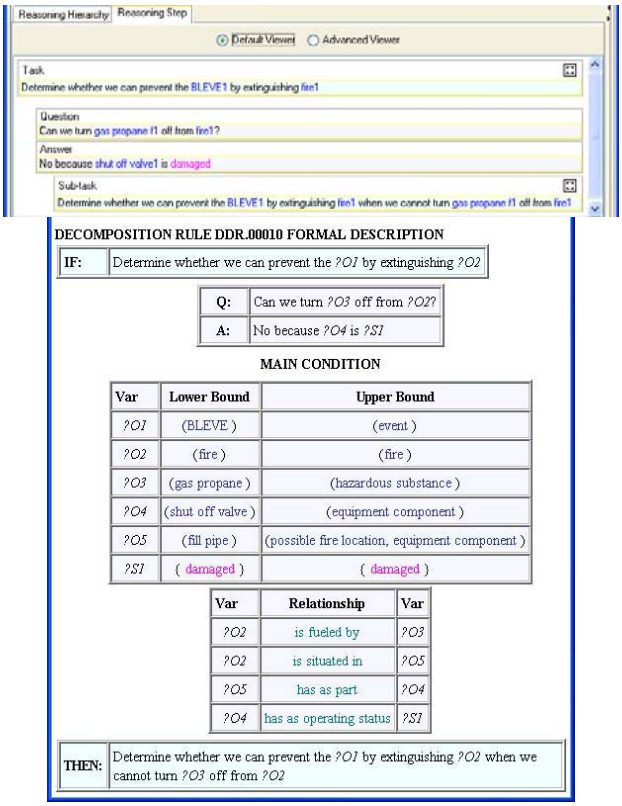

Figure 9. Example of a reduction step for an inference task (top) and the rule learned from it (bottom).

the relationships indicated in the relationship table, and so on. The plausible upper bound of the applicability condition is interpreted in a similar way, using the concepts from the Upper Bound table.

Rule learning is accomplished through a mixed-initiative process between the expert (who knows why the reduction is correct and can help the agent to understand this) and the Disciple-VE agent (which is able to generalize the task reduction example and its explanation into a general rule, by using the object ontology as a generalization language). The learning method is presented in Table 2 and is described in the following sections.

TABLE II.
THE LEARNING METHOD FOR TASK REDUCTION INFERENCE RULES

Let E be a reduction of a specific inference task T to one or several inference subtasks T_i, reduction taking place in state S_k

1. ***Example Understanding:*** Interact with the subject matter expert to understand the meaning of the question/answer pair from the example reduction E in terms of the objects and their features from the state S_k. These expressions represent the explanation EX of E.
2. ***Example Parameterization:*** Express the example E and its explanation EX into an equivalent IF-THEN rule R with the applicability condition IC, where each instance, number and string from the example and the explanation is parameterized to a variable. IC includes both the association of variables to instances or constants and the features from the explanation EX.
3. ***Rule Generalization:*** Generalize IC to a plausible version space condition of the rule R, where the plausible upper bound is the maximally general generalization of IC, and the plausible lower bound is the minimally general generalization of IC, both generalizations containing no instances and being based on the agent's object ontology.
4. ***Rule Analysis and Refinement:*** If the rule is determined to be incompletely learned then go to step 1 to identify additional explanation pieces for EX. Otherwise end the rule learning process.

B. Example Understanding

The question and its answer from the task reduction step represent the expert's reason (or explanation) for performing that reduction. Therefore understanding the example by Disciple-VE means understanding the meaning of the question/answer pair in terms of the concepts and features from the agent's ontology. This process is difficult for a learning agents that does not have much knowledge because the experts express themselves informally, using natural language and common sense, and often omit essential details that are considered obvious. The question/answer pair from the example in Fig. 9 is:

Can we turn gas propane f1 off from fire1? (7)
No because shut off valve1 is damaged.

We can expect a person to assume, without being told, that the reason we are considering turning the gas propane f1 off from fire1 is because it is fueling fire1. We can also expect the person to assume that we are considering shutting off valve1 because it is part of fill pipe1 with the propane. However, an automated agent is not able to make these assumptions and has to be helped to get an as complete understanding of the example as possible. For instance, a more complete explanation of the example from the top of Fig. 9 consists of the following facts:

fire1 is_fuelled_by gas_propane_f1 (8)
fire1 is_situated_in fill pipe1 has_as_part shutoff valve1
shutoff valve1 has_as_operating_status damaged
the value damaged is required

The process of identifying such explanation pieces is based on a communication protocol between the expert and Disciple-VE which takes into account that:

- It is easier for a human expert to understand sentences in the formal language of the agent, such as (8), than it is to produce such formal sentences.
- It is easier for the agent to generate formal sentences than it is to understand sentences in the natural language used by the expert, such as (7).

This protocol is implemented in the Explanation module of Disciple-VE, as explained in the following.

The subject matter expert cannot easily specify the explanation pieces in (8) because s/he is not a knowledge engineer. For instance, s/he would need to use the formal language of the agent. But this would not be enough, as the expert would also need to know the names of the potentially many thousands of concepts and features from the agent's ontology. However, the expert can understand the meaning of the above formal expressions. Therefore, the agent will hypothesize plausible meanings of the question-answer pair by using simple natural language processing, analogical reasoning with previously learned rules, and general heuristics, and will express them as explanation pieces, such as those in (8). In general, an ***explanation piece*** *is a relationship (or a relationship chain) involving instances, concepts, and constants from the task reduction step and from the knowledge base.*

The agent will propose the explanation pieces to the expert, ordered by their plausibility. Then the expert can select the explanation pieces that express approximately the same meaning as the question-answer pair, including the facts that were not explicitly stated, as discussed above. The expert may also help the agent to propose the right explanation pieces by providing hints, such as asking the agent to generate explanation pieces related to certain instances from the example.

The quality of the learned rule depends directly on the completeness of the found explanation. However, there is no requirement that the found explanation be complete and, in fact, this rarely occurs. The agent will continue to improve the rule while using it in reasoning, when it will be easier to discover the missing explanation pieces.

C. Example Parameterization

Example parameterization consists in transforming the example and its explanation into an equivalent IF-THEN rule, by replacing each instance or constant with a variable, and restricting the variables to those values, as illustrated in Table III.

TABLE III.
PARAMETERIZATION OF THE EXAMPLE IN FIG. 9 AND ITS EXPLANATION (8)

IF *the task is to*
Determine whether we can prevent the ?O1 by extinguishing ?O2

Q: Can we turn ?O3 off from ?O1?
A: No because ?O4 is ?S1

Condition IC

?O1	is	BLEVE1
?O2	is	fire1
	is_fuelled_by	?O3
	is_situated_in	?O5
?O3	is	gas propane f1
?O4	is	shutoff valve1
	has_as_operating_status	?S1
?O5	is	fill pipe1
	has_as_part	?O4
?S1	is	damaged

THEN
Determine whether we can prevent the ?O1 by extinguishing ?O2
when we cannot turn ?O3 off from ?O2

D. Rule Generalization

The expression in Table III is an instance of the general rule to be learned from the example in Fig. 9 and its explanation in (8). The next step in the rule learning process is to determine which are the values of the variables from the condition IC that lead to correct task reduction steps. That is, Disciple-VE has to learn the concept that represents the set of instances of the rule's variables for which the corresponding instantiation of the rule R is correct. We call this concept ***the applicability condition of the rule*** and Disciple-VE learns it by using a plausible version space approach.

First Disciple-VE generalizes the applicability condition IC to an initial plausible version space condition, as described in the following and illustrated in Fig. 10. *The* ***plausible upper bound condition*** *is obtained by replacing each variable value with its most general generalization, based on the object ontology.*

Let us consider the value fire1 of the variable ?O2. The most general concept from the object ontology which is more general than fire1 but less general than object is event. However, the possible values for ?O2 are restricted

by the features of fire1 identified as relevant as part of the explanation (8) of the example in Fig. 9. As indicated in Table III, ?O2 should have the features is_fuelled_by and is_situated_in. This means that the values of ?O2 have to be part of the domains of these features. Thus:

most general generalization(fire1) =
= {event} ∩ Domain(is_fuelled_by) ∩ Domain(is_situated_in)
= {event} ∩{fire}∩{object} = {fire}

Applying a similar procedure to each variable value from IC one obtains the plausible upper bound condition shown in Fig. 10.

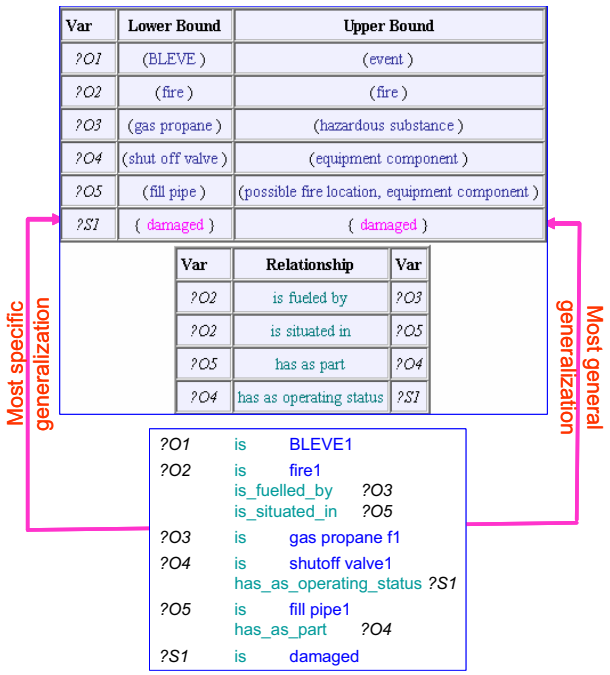

Figure 10. Generation of the initial plausible version space condition.

*The **plausible lower bound condition** is obtained by replacing each variable value with its least general generalization which is not an instance, based on the object ontology.* The procedure is similar with the one for obtaining the plausible upper bound condition. Therefore:

least general generalization(fire1) =
= {fire}∩ Domain(is_fuelled_by) ∩ Domain(is_situated_in) =
= {fire} ∩{fire}∩{object} = {fire}

The reason the lower bound cannot contain any instance is that the learned rule will be used by Disciple-VE in other scenarios where the instances from the current scenario (such as fire1) do not exist, and Disciple-VE would not know how to generalize them. On the other hand, we also do not claim that the concept to be learned is more general than the lower bound, as discussed in Sec. V.D and illustrated in Fig. 4.

Notice that the features from the explanation (8) significantly limit the size of the initial plausible version space condition and thus speed up the rule learning process. This is a type of explanation-based learning [28, 29] except that the knowledge base of Disciple-VE is incomplete and therefore rule learning requires additional examples and interaction with the expert.

E. Rule Analysis and Refinement

After the rule was generated, Disciple-VE analyzes it to determine whether it was learned from an incomplete explanation [30]. To illustrate, let us consider again the process of understanding the meaning of the question/answer pair in (7) in terms of the concepts and features from the agent's ontology. In the previous sections we have assumed that this process has led to the uncovering of implicit explanation pieces. However, this does not always happen. Therefore let us now assume that, instead of (8), the identified explanation pieces of the example are the following ones:

shutoff valve1 has_as_operating_status damaged (9)
The value damaged is required

In this case the learned rule is the one from Fig. 11.

*The variables from the IF task of a rule are called **input variables** because they are instantiated when the rule is invoked in problem solving. The other variables of the rule are called **output variables.***

During the problem solving process the output variables are instantiated by the agent with specific values that satisfy the rule's applicability condition. In a well-formed rule, the output variables need to be linked through explanation pieces to some of the input variables of the rule. Therefore one rule analysis method consists of determining whether there is any output variable which is not constrained by the input variables. For instance, in the case of the rule from Fig. 11, Disciple-VE determined that the variables ?O3, ?O4, and ?S1 are not constrained, and asks the expert to guide it to identify additional explanation pieces related to their corresponding values (i.e. gas propane f1, shutoff valve1 and damaged).

If the rule passes the structural analysis test, Disciple-VE determines the number of its instances in the current

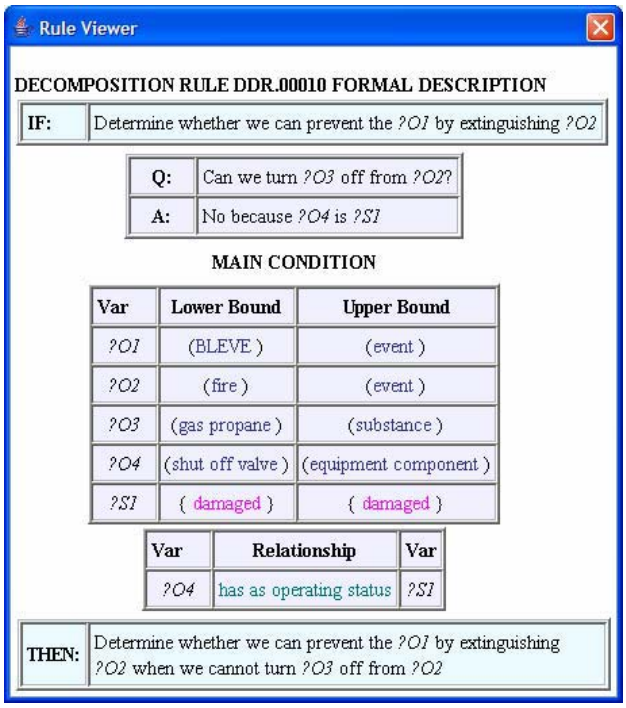

Figure 11. Rule learned from the example at the top of Fig. 9 and the explanation (9).

knowledge base and considers that the rule in incompletely learned if this number is above a pre-defined threshold. In such a case, the agent will attempt to identify which variables are the least constrained and will attempt to further constrain them by interacting with the expert to find additional explanation pieces.

Following such a process, Disciple-VE succeeds in learning a reasonable good rule from only one example and its explanation, a rule that may be used by Disciple in the planning process. The plausible upper bound condition of the rule allows it to apply to situations that are analogous with the one from which the rule was learned. If the expert judges this application as correct, then this represents a new positive example of the rule, and the plausible lower bound condition is generalized to cover it. Otherwise, the agent will interact with the expert to find an explanation of why the application is incorrect, and will specialize the rule's conditions appropriately. Rule refinement could lead to a complex task reduction rule, with except-when conditions which should not be satisfied in order for the rule to be applicable.

X. Teaching an Agent to Perform Planning Tasks

A. Why Learning Planning Rules is Difficult

Fig. 12 compares the learning of inference rules with the learning of planning rules. The left hand side of Fig. 12 shows an inference step and a planning step, while the right hand side shows the rules that would be learned from these steps. In the case of an inference step, Disciple-VE learns a rule by generalizing the expressions from the examples to patterns, and by generating a plausible version space for the applicability condition of the rule.

The learning of the planning rule is much more complex, not just because it involves the learning of several applicability conditions, but mainly because these conditions have to be learned in different states of the world. Indeed, Condition 1g and Condition 2g are learned in the state S_1, but Condition 3g has to be learned in the state S_3. However, the state S_3 is only known after the entire reduction tree for Planning Task 2 has been developed. What this means is that Disciple-VE would start learning the rule in the state S_1, will than continue with the planning and inference corresponding to the subtree of Planning Task 2, and only after that can resume and finalize the learning of the rule. But this is impractical for two main reasons. First, it leads to the starting of learning many complex planning rules, with the associated management of temporary representations for these rule fragments. Second, these incompletely learned rules cannot be used in problem solving. Thus, in the case of a planning tree that contains recursive applications of a task reduction step Disciple would start learning a new rule for each application, although these rules will end up being identical.

B. Learning a Set of Correlated Planning Rules

The main source of difficulty for learning a planning rule from the planning example in Fig. 12 is the need to first develop the entire planning tree for Planning Task 2. We have discussed a similar difficulty in Sec. VII, in the context of modeling expert's planning process. In that case the expert could not specify the reduction of Planning Task 1 into Planning Task 2 and Planning Task 3 before completing the entire planning for Planning Task 2. The solution found to that problem was to introduce the notion of abstract task. This notion will also help overcome the difficulty of learning planning rules, as will be explained in the following.

Rather than learning a single complex planning rule from a task reduction example, Disciple-VE will learn a set of simpler planning rules that share common variables, as illustrated in the right part of Fig. 13. These rules will not be learned all at once, but in the sequence indicated in Fig. 14. This sequence corresponds to the sequence of modeling operations for the subtree of Planning Task 1, as discussed in Sec. VII. First the expert asks himself/herself a question related to how to reduce Planning Task 1. The answer guides him/her to reduce this task to two abstract tasks. From this reduction the agent learns a planning task reduction rule (see Fig. 14a), by using the

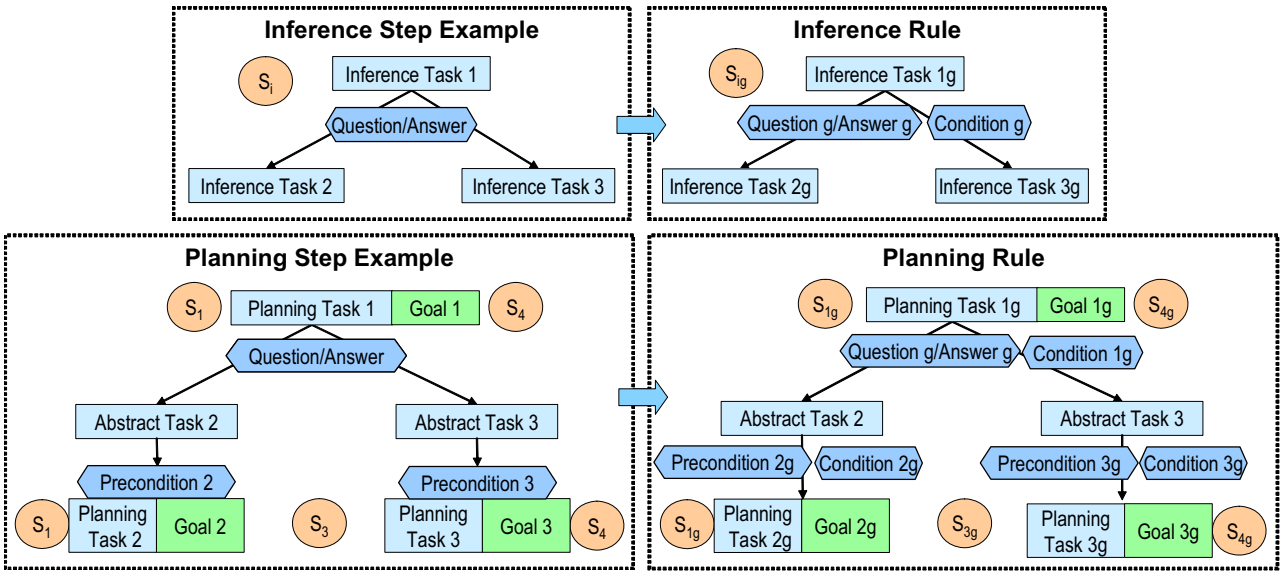

Figure 12. Learning Inference Rules Versus Learning Planning Rules.

method described in Sec. X.D. Next the expert reduces Abstract Task 2 to Planning Task 2 and the agent learns a task concretion rule (see Fig. 14b), by using the method described in Sec. X.E. After that the expert continues with specifying the reduction tree corresponding to Planning Task 2 and the agent learns rules from the specified planning step, as indicated above. During the development of this planning tree, the agent may apply the above rules, if their conditions are satisfied, and may refine them based on expert's feedback. After the entire subtree corresponding to Planning Task 2 was developed, the agent can learn the Goal Mapping Rule corresponding to Goal 2, as described in Sec. X.D. The learning of the concretion rule for Abstract Task 3 and of the goal mapping rule for Goal 3 is done as described above. After that Disciple-VE learns the goal synthesis rule corresponding to Goal 1, as described in Sec. X.D.

The above illustration corresponds to a reduction of a planning task into planning subtasks. However, a planning task can also be reduced to elementary actions, as illustrated at the bottom part of Fig. 13. In this case Disciple-VE will learn more complex action concretion rules instead of task concretion rules, as discussed in Sec. X.F.

In the following sections we will present the learning methods mentioned above.

C. The Learning Problem and Method for a Set of Correlated Planning Rules

The problem of learning a set of correlated planning rules is presented in Table IV, and the corresponding learning method is presented in Table V. We will illustrate them by using the top task reduction from Fig. 6.

TABLE IV.
THE LEARNING PROBLEM FOR CORRELATED PLANNING RULES

Given:
- A sequence of reduction and synthesis steps SE that indicate how a specific planning task is reduced to its immediate specific subtasks and/or actions, and how its goal is synthesized from their goals/effects.
- A knowledge base that includes an ontology and a set of rules.
- A subject matter expert who understands why the given planning steps are correct and may answer the agent's questions.

Determine:
- A set of reduction, concretion, goal and/or action rules SR which share a common space of variables, each rule being a generalization of an example step from SE.
- An extended object ontology (if needed for rule learning).

D. Learning a Correlated Planning Task Reduction Rule

The method for learning a correlated planning reduction rule in presented in Table VI. This method is similar

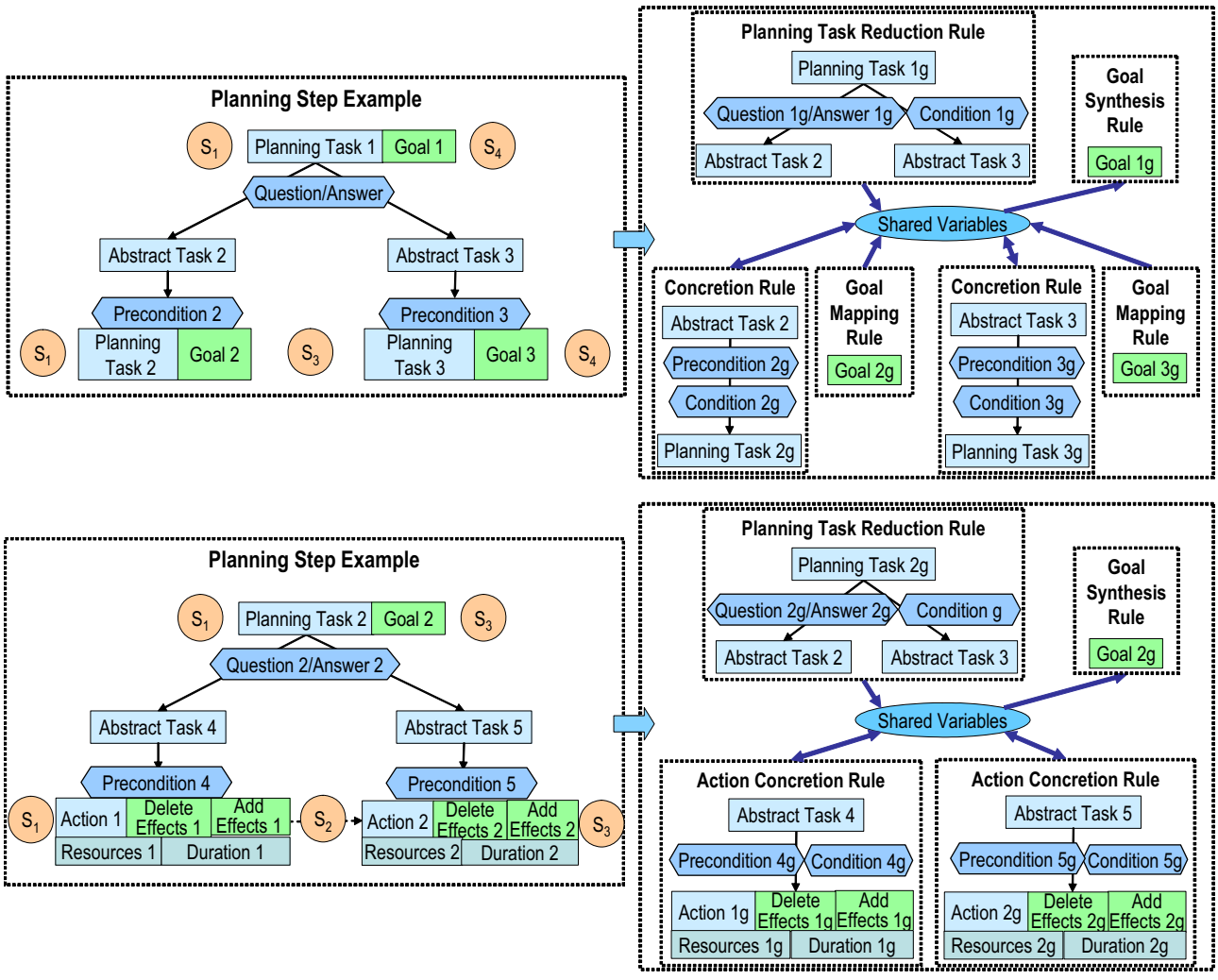

Figure 13. Learning Correlated Planning Rules.

TABLE V.
THE LEARNING METHOD FOR CORRELATED PLANNING RULES

Let SE be a sequence of reduction and synthesis steps that indicate how a specific planning task T is reduced to its immediate specific subtasks and/or actions, and how its goal is synthesized from their goals/effects.

1. Initialize the set V of shared variables and their values in SE $V \leftarrow \Phi$

2. Learn a Planning Task Reduction Rule from the reduction of T to the abstract tasks AT_i and update the set V (by using the method described in Table VI and Sec. X.D)

3. For each abstract task AT_i do

If AT_i is reduced to a concrete Task T_i

Then 3.1. Learn a Planning Task Concretion Rule and update set V (by using the method described in Sec. X.E)

3.2. Develop the entire subtree of T_i (this may lead to learning of new rules by using the methods from Tables II and V)

3.3. Learn the Goal Mapping Rule for T_i (by using the method described in Sec. X.D)

Else if AT_i is reduced to an elementary action A_i

Then 3.1. Learn an Action Concretion Rule and update the set V (by using the method described in Sec. X.F)

4. Learn the Goal Synthesis Rule for T (by using the method described in Sec. X.D.

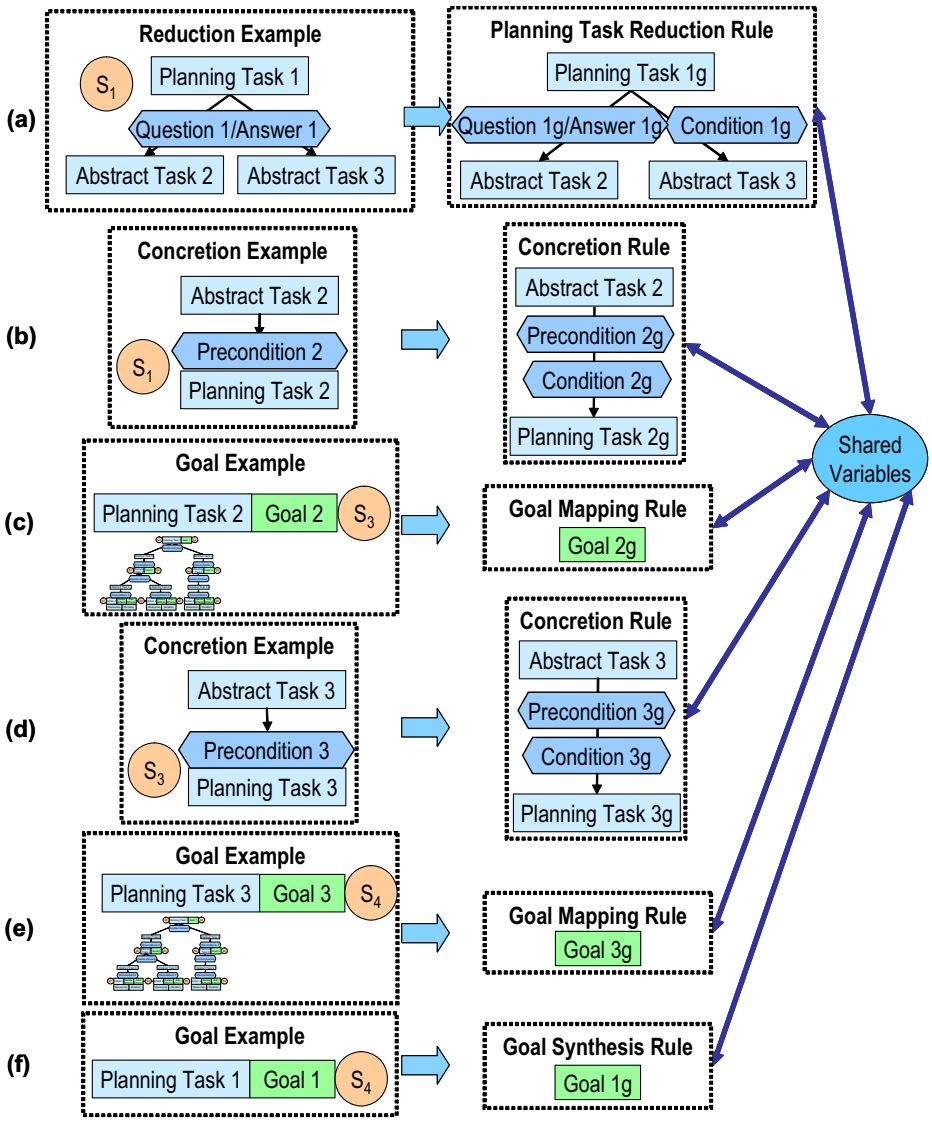

Figure 14. The sequence of learning correlated planning rules.

to the method of learning an inference rule presented in Table II and Sec. IX, except for the addition of Step 3 in Table VI which adds to the set V the variables from the learned rule and their values in the example.

To illustrate the method in Table VI, let us consider the top-level reduction from Fig. 6. In that reduction the top level task is reduced to 5 abstract tasks. The reduction is justified by the following Question/Answer pair:

What king of incident is Gainsville incident?
Gainsville incident is a major fire emergency
because it involves a large propane tank on fire.

As part of example understanding, Disciple-VE will interact with the expert to find the following explanation pieces which represents an approximation of the meaning of the Question/Answer pair in the current world state:

Gainsville incident instance_of major fire emergency
Gainsville incident is_caused_by fire1 has_as_size large
fire1 is_impinging_on propane tank1 instance_of
propane tank

Continuing with the steps from Table VI Disciple-VE will learn the rule from the left-hand side pane of Fig. 15. The list of shared variables is shown in the right hand side of this pane. The right hand side of the pane shows also the goal produced by the Goal synthesis rule. This rule generalizes the expression representing the goal associated with the IF task by replacing its instances and constants with the corresponding variables from the list of shared variables. Similarly, the Goal mapping rule generalizes the goals of the THEN tasks.

E. Learning Correlated Planning Task Concretion Rules

The method of learning a correlated planning task concretion rule is similar to the method of learning a correlated planning reduction rule presented in Table VI and Sec. X.D. To illustrate it, let us consider again the reduction in the left side of Fig. 6. The Sub-Task(3) pane includes a concretion step which is shown again in Table VII. The rule learned from this concretion example is shown in the right pane of Fig. 15.

As part of ***Example understanding*** (see Table VI), what needs to be understood are the preconditions of the concre-

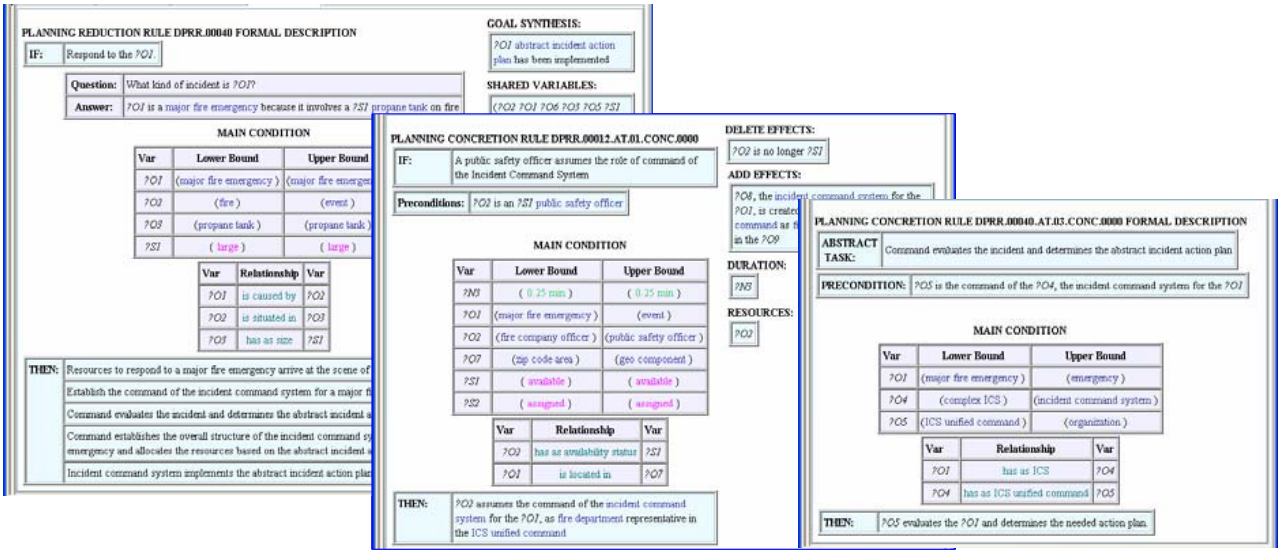

Figure 15. Planning task reduction rule (left), action concretion rule (middle) and task concretion rule (right).

tion step. The approximation of their meaning is:

Gainsville incident has_as_ICS Gainsville ICS
has_as_unified_command Gainsville command

The ***Rule Analysis and Refinement*** step takes the value of the set V into account to determine the unlinked output variables (see Sec. IX.E). In particular, an output variable from the concrete task does not need to be linked to input variables if it is part of the input value of V.

F. Learning a Correlated Action Concretion Rule

If the reduction of a planning tasks includes actions, then Disciple-VE learns also correlated action concretion rules, as illustrated at the bottom part of Fig. 13. The learning method is similar to that for learning a correlated task concretion rule, except that the resulting rule has additional action components such as Delete Effects, Add Effects, Resources and Duration.

TABLE VI.
THE LEARNING METHOD FOR A CORRELATED PLANNING TASK REDUCTION RULE

Let E be a reduction of a specific planning task T to one or several abstract tasks AT_i, reduction taking place in state S_k, and let V be the set of shared variables and their values.

1. ***Example Understanding:*** Interact with the expert to understand the meaning of the question/answer pair from the example reduction E in terms of the objects and their features from the state S_k. These expressions represent the explanation EX of the example E.
2. ***Example Parameterization:*** Express the example E and its explanation EX into an equivalent IF-THEN rule R with the applicability condition IC, where each instance, number and string from the example and the explanation is parameterized to a variable. IC includes both the association of variables to instances or constants and the features from the explanation EX.
3. ***Updating of Shared Variables and Values:*** Add to the set V the new variable and their values from the condition IC.
4. ***Rule Generalization:*** Generalize IC to a plausible version space condition R, where the plausible upper bound is maximally general generalization of IC, and the plausible lower bound is the minimally general generalization of IC, both generalizations containing no instances and being based on the agent's object ontology.
5. ***Rule Analysis and Refinement:*** If the rule is determined to be incompletely learned then go to step 1 to identify additional explanation pieces for EX. Otherwise end the rule learning process.

Let us consider the abstract task from the bottom pane in Fig. 6 and its concretion. The action concretion rule learned from this example is shown in the middle pane of Fig. 15.

TABLE VII.
A TASK CONCRETION EXAMPLE

Abstract: Command evaluates the incident and determines the abstract incident action plan.
Preconditions: Gainsville command is the command of the Gainsville ICS, the incident command system for the Gainsville incident.
Task: Gainsville command evaluates the Gainsville incident and determines the needed incident action plan.

XI. THE VIRTUAL EXPERTS LIBRARY

A. The Organization of the VE Library

The previous sections have presented the capability of a Disciple-VE agent shell to rapidly acquire planning expertise from a subject matter expert. This capability makes possible the development of a wide range of planning agents that can collaborate in performing complex tasks. These agents are maintained into the VE Library where their knowledge bases (KBs) are hierarchically organized, as illustrated in Fig. 16. In this illustration there are four expertise domains, D1, D2, D3, and D4, and nine virtual experts, each associated with a KB from the bottom of the hierarchy. Each virtual expert engine VE is a customization of the Disciple-VE shell. The three leftmost virtual experts are experts in the domain D1 with different levels of expertise: basic, intermediary and advanced. In addition to their specific KBs (e.g. KB-B1), they all inherit general knowledge about the domain D1, knowledge represented in KB-D1. They also inherit knowledge from the higher level KBs KB-12 and KB-0. These higher level KBs contain general knowledge, useful to many agents, such as ontologies for units of measure, time, and space.

Traditional knowledge engineering practice builds each KB from scratch, with no knowledge reuse, despite the fact that this is a very time-consuming, difficult and error-prone process [6-10]. On the contrary, the hierarchy

of KBs from the VE Library offers a practical solution to the problem of knowledge reuse, speeding up the process of building a new virtual expert. Consider, for example, developing a new virtual expert for the D3 domain. This expert will already start with a KB composed of KB-D3, KB-34, and KB-0. Thus, the VE Library can also be regarded as a knowledge repository for the new virtual experts to be developed.

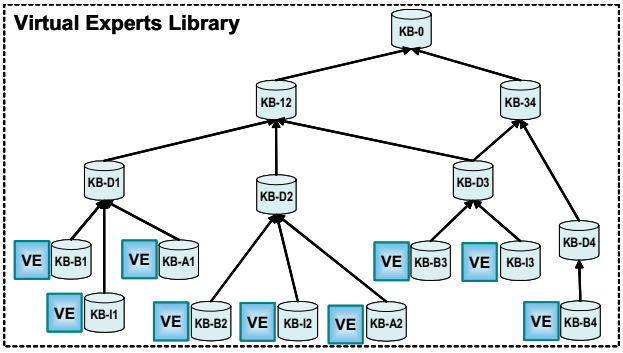

Figure 16. The organization of the VE Library.

B. Knowledge Base Development and Maintenance

The updating of each KB from the hierarchical repository (e.g. KB-12) is the responsibility of a team consisting of a knowledge engineer and one or several subject matter experts. They use the specialized browsers and editors of the Disciple-VE shell. The left hand side of Fig. 17 shows the interface of the Object Browser which displays the objects in a tree structure. The objects that are inherited from an upper level KB (such as information measure or length measure) are displayed in a gray background. The right hand side of Fig. 17 shows the interface of the Object Viewer, which displays additional information about the object selected in the Object Browser (e.g. fire engine company E501), such as its direct super-concepts and its features. The top part of Fig. 3 shows the interface of the Hierarchical Browser which displays the hierarchical relationships between objects or features in a graph structure. The bottom part of Fig. 3 shows the interface of the Association Browser which displays an object and its relationships with other objects. Additional tools include the Object Editor, the Feature Editor, and the Rule Editor.

To allow the KBs from the hierarchy to be updated and extended separately, the Disciple-VPT system maintains multiple versions for each KB. Let us assume that each KB from Fig. 16 has the version 1.0. Let us further assume that the management team for KB-0 decides to make some changes to this KB which contains units of measure. For instance, the team decides to include the metric units, to rename gallon into US gallon, and to add UK gallon. As a result, the team creates version 2.0 of KB-0. However, the other knowledge bases from the library (e.g. KB-12) still refer to version 1.0 of KB-0. The management team for KB-12 is informed that a higher version of KB-0 is available. At this point the team can decide whether it wants to create a new version of KB-12 that inherits knowledge from version 2.0 of KB-0. The KB update process uses the KB updating tool of Disciple-VE. This tool creates version 2.0 of KB-12 by importing the knowledge from version 1.0 of KB-12, in the context of version 2.0 of KB-0. Even though the version 2.0 of KB-12 has been created, Disciple-VPT still maintains KB-0 version 1.0 and KB-12 version 1.0, because these versions are used by KB-D1 version 1.0 and by other KBs from the repository. The management team for KB-D1 may now decide whether it wants to upgrade KB-D1 to the new versions of its upper level KBs, and so on. Because of the version system, each KB from the library maintains, in addition to its version, the versions of the other KBs from which it inherits knowledge.

Another important knowledge management functionality offered by Disciple-VPT is that of splitting a KB into two parts, a more general one and a more specific one. This allows a KB developer to first build a large KB and then to split it and create a hierarchy of KBs.

C. Organization of an Agent's Knowledge Base

When a virtual expert is extracted from the VE Library and introduced into a VE Team (see Fig. 1), all the KBs from which it inherits knowledge are merged into a shared KB in order to increase the performance of the agent. Let us consider the Intermediate agent from the domain D3 (see Fig. 16). In this case KB-D3, KB-12, KB-34

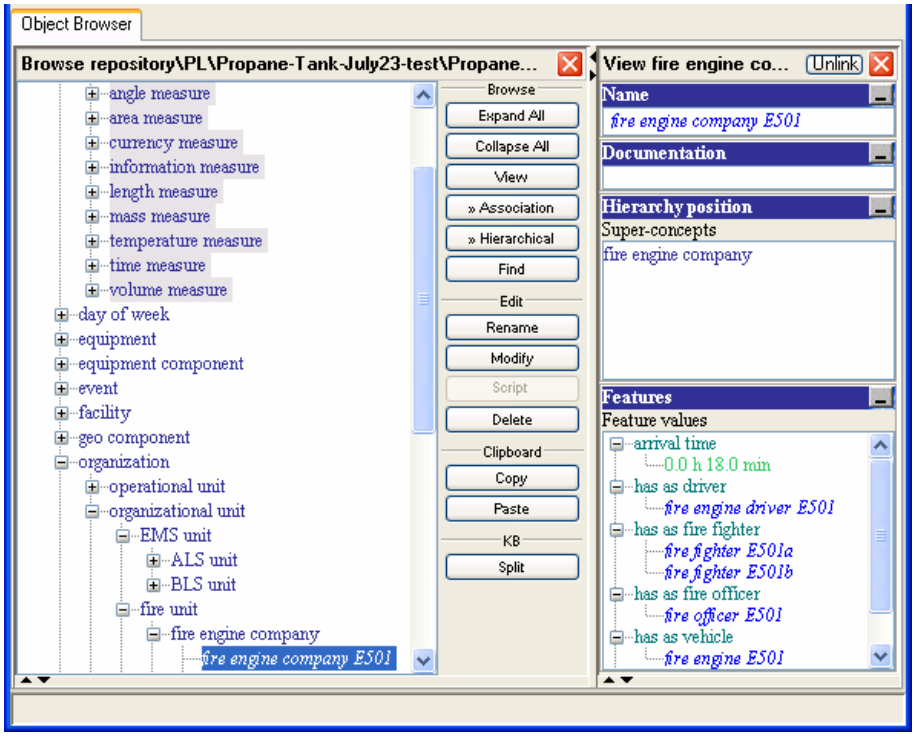

Figure 17. The interface of the Object Browser and Object Viewer.

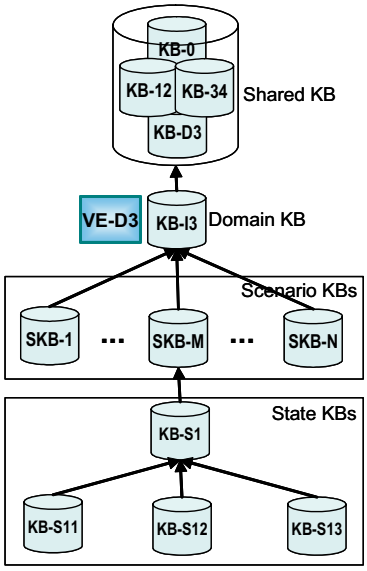

Figure 18. The organization of an agent's knowledge base during planning.

and KB-0 are all merged into the Shared KB of this agent. As a conesquence, the structure of the KBs of this agent during planning is the one from Fig. 18. Notice that, in addition to the Shared KB, there are three other types of KBs, Domain KB, Scenario KB, and State KB, all hierarchically organized. Domain KB is the KB of this Intermediate agent from the domain D3, knowledge independent of any particular scenario. Each scenario is represented into a different KB called Scenario KB. For example, there would be a Scenario KB for the propane tank of fire scenario described in Sec. III, and a different Scenario KB for a scenario involving red-fuming nitric acid spilling from a truck parked near a residential area [2, 31]. Moreover, under each scenario KB there is a hierarchy of State KBs. KB-S1 represents the state obtained from SKB-M after the execution of an action which had delete and/or add effects. As additional actions are simulated during planning, their delete and add effects change the state of the world. KB-S11, KB-S12, and KB-S13 are the states corresponding to three alternative actions. The entire description of the state corresponding to KB-S11 is obtained by considering the delete and add effects in the states KB-S11 and KB-S1, and the facts in the scenario SKB-M.

XII. Multi-Domain Collaborative Planning

Each virtual agent from the VE Library is expert in a certain expertise domain. However, these expertise domains are not disjoint, but overlapping, as illustrated in Fig. 19. In this illustration the planning task T_m belongs only to D_2 and can only be performed by a virtual expert from that domain. T_i is a task common to D_1 and D_2 and can, in principle, be performed either by a virtual expert in D_1 or by a virtual expert in D_2. In general, a virtual expert will only cover a part of a given expertise domain, depending on its level of expertise. For instance, the virtual expert library illustrated in Fig. 16 includes three virtual experts from the domain D_1, a basic one, an intermediate one, and an advanced one, each covering an increasingly larger portion of the domain. Therefore, whether a specific virtual expert from the domain D_2 can generate a plan for T_m, and the actual plan generated, depend on its level of expertise.

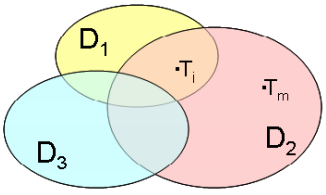

Figure 19. Sample coverage of expertise domains.

A virtual expert has partial knowledge about its ability to generate plans for a given task, knowledge that is improved through learning. For instance, the virtual expert knows that it may be able to generate plans for a given task instantiation because that task belongs to its expertise domain, or because it was able to solve other instantiations of that task in the past. Similarly, it knows when a task does not belong to its area of expertise. The virtual experts, however, do not have predefined knowledge about the problem solving capabilities of the other experts from a VE Team, or from the VE Library. This is a very important feature of Disciple-VPT that facilitates the addition of new agents to the library, or the improvement of the existing agents, because this will not require taking into account the knowledge of the other agents.

The task reduction paradigm facilitates the development of plans by cooperating virtual experts, where plans corresponding to different subtasks of a complex task may be generated by different agents. This multi-agent planning process is driven by an auction mechanism that may apply several strategies. For instance, the agents can compete for solving the current task based on their prior knowledge on their ability to solve that task. Alternatively, the agents may actually attempt to solve the task before they bid on it.

XIII. Basic Virtual Experts

We have developed two basic virtual experts, a fire department operations expert, and an emergency management expert. The development of these virtual experts was guided by a toxic substance leaking scenario described in [2] and by the propane tank on fire scenario described in Sec. III.

First we have worked with a subject matter expert to model the plan generation process for these two scenarios by using the task reduction paradigm, as illustrated in Fig. 6. Besides the development of the two reasoning trees, another result of this modeling process was the development of the modeling methodology presented in Sec. VII. Yet another result of this modeling process was the identification of the object concepts that need to be present in Disciple's ontology so that it can perform this type of reasoning. Based on this specification of the object ontology, and by using the ontology development modules of Disciple-VE, we have developed an object ontology consisting of 410 concepts, 172 feature definitions, 319 generic instances, and 944 facts. Fragments of this ontology were presented in Fig. 2 and Fig. 3.

Although we have worked with both scenarios mentioned above to develop the modeling trees and the ontology, the teaching of the fire expert and of the emergency management expert was only based on the propane tank on fire scenario. As a result of the teaching process, the virtual fire expert learned 81 planning rules, and the virtual emergency management expert learned 47 planning rules. The current versions of the two developed virtual experts share the object ontology. This ontology will be split into shared and private ontologies in the next versions of these agents.

XIV. EVALUATION

To evaluate the Disciple-VPT system we have developed a ***scenario pattern*** based on the scenario from Sec. III. Then we have asked a fire department operations expert and an emergency management expert to define a specific scenario based on this pattern, by providing the missing elements. For example, the two experts decided on the day and time of the incident, the position of the fire with respect to the propane tank, the estimated amount of propane in the tank, the available resources (e.g. fire engine companies, truck engine companies, hazmat units, police personnel, county personnel, etc.) and when are they arriving at the scene of the incident. After that, both the team of human experts and Disciple-VPT independently generated plans to respond to this situation.

The plan generated by Disciple-VPT consisted of 89 partially ordered elementary actions. A fragment of this plan is shown in Table VIII. This plan was evaluated by the above mentioned experts and by the expert with whom we have developed the agents. Then each expert has filled-in a questionnaire. The questionnaire included statements about various characteristics of the generated plan and about the Disciple-VPT system. The experts were asked to indicate whether they strongly agree (SA), agree (A), are neutral (N), disagree (D) or strongly disagree (SD) with these statements.

The top part of Table IX presents some of the results of the evaluation of the plan generated by Disciple-VPT, which was considered good and easy to understand by the human experts. One of the experts disagreed with the way Disciple-VPT ordered some of the actions. However, this reflected a disagreement between the evaluating experts and not a planning error, the generated order being that taught by the expert who instructed Disciple-VPT.

The evaluation of the hierarchical plan generation

TABLE VIII.
FRAGMENT OF A PLAN GENERATED BY DISCIPLE-VPT

Id	Action	Result	Resources	Sart Time	Duration
1	Suburbane Propane Company facility manager, a person, arrives at the scene of the Gainsville incident	**Add:** Suburbane Propane Company facility manager arrived at the scene and is available to take required actions	Suburbane Propane Company facility manager	0.0 s	1.0 min 0.0 s
2	ALS unit M504, an ALS unit, arrives at the scene of the Gainsville incident	**Add:** ALS unit M504 arrived at the scene and is available to take required actions	ALS unit M504, paramedic 504a, and paramedic 504b	0.0 s	5.0 min 0.0 s
3	fire engine company E504, a fire engine company, arrives at the scene of the Gainsville incident	**Add:** fire engine company E504 arrived at the scene and is available to take required actions	fire engine driver E504, fire fighter E504b, fire engine company E504, fire engine E504, deluge nozzle E504, water hose E504, fire officer E504, and fire fighter E504a	0.0 s	5.0 min 0.0 s
	…	…	…	…	…
48	fire officer E504 assumes the command of the incident command system for the Gainsville incident, as fire department representative in the ICS unified command	**Delete:** fire officer E504 is no longer available **Add:** Gainsville ICS, the incident command system for the Gainsville incident, is created and fire officer E504 assumes ICS command as fire department representative in the Gainsville command	fire officer E504	5.0 min 0.0 s	15.0 s
49	Gainsville command evaluates the situation created by the Gainsville incident	**Add:** overpressure situation with danger of BLEVE in propane tank1 is caused by fire1	Gainsville command	5.0 min 15.0 s	30.0 s
50	Gainsville command determines the abstract incident action plan for overpressure situation	**Add:** The plan is to apply cooling water, to evacuate people from 1.0 mi around propane tank1, to perform traffic control, perimeter control, and to establish rapid intervention task forces	Gainsville command	5.0 min 45.0 s	30.0 s
	…	…	…	…	…
54	fire engine company E504 sets up water hose E504 to apply water to propane tank1	**Delete:** water hose E504 is available **Add:** water hose E504 is assigned	fire engine company E504, fire engine driver E504, fire fighter E504b, fire fighter E504a, water hose E504, and fire engine E504	8.0 min 30.0 s	3.0 min 0.0 s
55	fire engine company E525 drops off deluge nozzle E525 for fire engine company E504	**Delete:** deluge nozzle E525 that belongs to fire engine E525 is no longer available **Add:** deluge nozzle E525 is assigned	fire fighter E525a, fire officer E525, deluge nozzle E525, fire engine driver E525, and fire fighter E525b	8.0 min 30.0 s	2.0 min 0.0 s
	…	…	…	…	…
63	fire engine company E525 establishes continuous water supply from fire hydrant1 for fire engine company E504	**Add:** fire hydrant1 is assigned to fire engine company E504	fire fighter E525a, fire fighter E525b, fire officer E525, fire engine driver E525, and fire engine company E525	14.0 min 30.0 s	7.0 min 0.0 s
	…	…	…	…	…

process (see the middle of Table IX) shows that it significantly improves the understandability of this process. Finally the experts agreed or strongly agreed that Disciple-VPT has many potential applications in emergency response planning, from developing exercises for training, to actual training, and to its use as planning assistant.

A limitation of this evaluation is that it is based on the opinion of only 3 experts. More experts are needed in order for the results to have statistical significance.

XV. Contributions and Future Directions

This paper presented a major extension of the Disciple theory and methodology for the development of knowledge-based agents by subject matter experts, with limited assistance from knowledge engineers.

First we have extended the Disciple approach to allow the development of complex HTN planning agents that can be taught their planning knowledge, rather then having it defined by a knowledge engineer. This is a new and very powerful capability that is not present in current action planning systems [1, 18-20]. This capability was made possible by several major developments of the Disciple approach. For instance, we have significantly extended the knowledge representation and management of a Disciple agent by introducing new types of knowledge that are characteristic to planning systems, such as planning tasks and actions (with preconditions, effects, goal, duration, and resources) and new types of rules (e.g. planning tasks reduction rules, concretion rules, action rules, goal synthesis rules). We have introduced state knowledge bases and have developed the ability to manage the evolution of the states in planning. We have developed a modeling language and a set of guidelines that help subject matter experts express their planning process. We have developed an integrated set of learning methods for planning, allowing the agent to learn general planning knowledge from a single planning example formulated by the expert.

TABLE IX.
Evaluation Results

Generated Plan	SA	A	N	D	SD
The generated plan is easy to understand.		3			
The generated plan consists of a good sequence of actions.		2		1	
The objectives of the actions are clear from the generated plan.		3			
The generated plan has a good management of the resources.	1	1	1		
The level of detail in the plan is appropriate.		3			

Hierarchical Plan Generation Process	SA	A	N	D	SD
The hierarchical task-reduction structure makes the logic of the plan clear.	2	1			
The hierarchical task-reduction structure makes the goals of the plan clear.	1	2			
The questions, the answers, and the preconditions help understand the logic of the plan generation process.	2	1			
The preconditions, effects, duration, and resources, make the specification of the plan's actions clear.	1	2			
The hierarchical task-reduction structure may be used to teach new persons how to plan.	3				

Usability of Disciple-VPT	SA	A	N	D	SD
Disciple-VPT could be used in developing and carrying out exercises for emergency response planning.	2	1			
Disciple-VPT could be used for training the personnel for emergency response planning.	2	1			
Disciple-VPT could be used as an assistant to typical users, guiding them how to respond to an emergency situation.		3			

A second result is the development of an integrated approach to planning and inference, both processes being based on the task reduction paradigm. This improves the power of the planning systems that can now include complex inference trees. It also improves the efficiency of the planning process because some of the planning operations can be performed as part of a much more efficient inference process that does not require a simulation of the change of the state of the world.

A third result is the development and implementation of the concept of library of virtual experts. This required the development of methods for the management of a hierarchical knowledge repository. The hierarchical organization of the knowledge bases of the virtual experts also serves as a knowledge repository that speeds-up the development of new virtual experts that can reuse the knowledge bases from the upper levels of this hierarchy.

A fourth result is the development of the multi-domain architecture of Disciple-VPT which extends the applicability of the current expert systems to problems whose solutions require knowledge of more than one domain [31].

A fifth result is the development of two basic virtual experts, a basic fire expert and a basic emergency management expert, that can collaborate to develop plans of actions that are beyond their individual capabilities.

Finally, a sixth result is the development of an approach and system that has high potential for supporting a wide range of training and planning activities.

Future research will involve the development of additional agents for the VE Library, and of the Disciple-VPT system for actual use in training personnel for emergency response planning, and for other areas.

Acknowledgment

This research was inspired and initiated by Dr. Susan Durham and was performed in the Learning Agents Center. The research of the Learning Agents Center is sponsored by several U.S. government agencies including the Air Force Research Laboratory (FA8750-04-1-0527), the Air Force Office of Scientific Research (F9550-07-1-0268), the National Science Foundation (0610743), and the Army War College. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

References

[1] Ghallab M., Nau D., and Traverso P., *Automatic Planning: Theory and Practice,* Morgan Kaufmann, 2004.

[2] Tecuci G., Boicu M., Hajduk T., Marcu D., Barbulescu M., Boicu C., Le V., A Tool for Training and Assistance in Emergency Response Planning, in *Proc. of the Hawaii Int. Conf. on System Sciences,* HICSS40, Hawaii, Jan.3-6, 2007
[3] Department of Homeland Security, National Response Plan, 2004, www.dhs.gov/dhspublic/interapp/editorial/editorial_0566.xml (accessed on 25 July, 2007).
[4] Tecuci G., Boicu M., Bowman M., and Marcu D., with a commentary by Burke M., An Innovative Application from the DARPA Knowledge Bases Programs: Rapid Development of a High Performance Knowledge Base for Course of Action Critiquing. *AI Magazine,* 22, 2:43-61, AAAI Press, Menlo Park, CA, 2001.
[5] Tecuci G., Boicu M., Marcu D., Stanescu B., Boicu C., Comello J., Training and Using Disciple Agents: A Case Study in the Military Center of Gravity Analysis Domain, in *AI Magazine,* 24, 4:51-68, AAAI Press, CA, 2002.
[6] Buchanan, B. G. and Wilkins, D. C. (eds.), *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems,* Morgan Kaufmann, San Mateo, CA, 1993.
[7] Durkin J., *Expert Systems: Design and Development,* Prentice-Hall, 1994.
[8] Awad, E. M., *Building Expert Systems: Principles, Procedures, and Applications,* West Publishing Company, 1996.
[9] Jackson P., *Introduction to Expert Systems,* Addison-Wesley, Essex, England, 1999.
[10] Awad, E., and Ghaziri M. H., *Knowledge Management,* Pearson Education Inc., Upper Saddle River, NJ, 2004.
[11] Federal Emergency Management Agency, National Incident Management System, http://www.fema.gov/emergency/nims/index.shtm (accessed on 25 July 2007).
[12] Tecuci, G., Disciple: A Theory, Methodology and System for Learning Expert Knowledge, *Thèse de Docteur en Science,* University of Paris-South, 1988.
[13] Tecuci, G., *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. London, England: Academic Press, 1998.
[14] Boicu, M., *Modeling and Learning with Incomplete Knowledge,* PhD Dissertation in Information Technology, Learning Agents Laboratory, School of Information Technology and Engineering, George Mason University, 2002.
[15] Nilsson, N.J., *Problem Solving Methods in Artificial Intelligence,* NY: McGraw-Hill, 1971.
[16] Powell G.M. and Schmidt C.F., A First-order Computational Model of Human Operational Planning, *CECOM-TR-01-8,* US Army CECOM, Fort Monmouth, NJ, 1988.
[17] Durham S., Product-Centered Approach to Information Fusion, *AFOSR Forum on Information Fusion,* Arlington, VA, 18-20 October, 2000.
[18] Tate, A., Generating Project Networks, In *Proceedings of IJCAI-77*, Massachusetts, pp. 888-893, 1977.
[19] Allen J., Hendler J., and Tate A. (eds), *Readings in Planning,* Morgan Kaufmann Publishers, 1990.
[20] Nau, D., Au, T., Ilghami, O., Kuter, U., Murdock, J., Wu, D., Yaman, F., SHOP2: An HTN planning system, *Journal of Artificial Intelligence Research*, 20, pp. 379–404, 2003.
[21] Fensel D., *Ontologies: A Silver Bullet for Knowledge Management,* Springer-Verlag, Berlin, 2000.
[22] Stanescu B., Boicu C., Balan G., Barbulescu M., Boicu M., Tecuci G., Ontologies for Learning Agents: Problems, Solutions and Directions, in *Proc. IJCAI-03 Workshop on Ontologies and Distributed Systems,* pp 75-82, Acapulco, Mexico, August, AAAI Press, Menlo Park, CA, 2003.
[23] Tecuci G., Boicu M., Cox M.T., Seven Aspects of Mixed-Initiative Reasoning, in *AI Magazine*, 28, 2:11-18, AAAI Press, Menlo Park, CA, 2007.
[24] Tecuci G., Boicu M., Cox M.T. (eds.), *AI Magazine*: *Special Issue on Mixed-Initiative Assistants,* 28, 2, AAAI Press, Menlo Park, CA, 2007.
[25] Michalski R.S. and Tecuci G. (eds.), *Machine Learning: A Multistrategy Approach,* vol. IV, 782 pages, Morgan Kaufmann, San Mateo, 1994.
[26] Mitchell, T.M., Version Spaces: an Approach to Concept Learning, *Doctoral Dissertation,* Stanford Univ., 1978.
[27] Bowman, M., *A Methodology for Modeling Expert Knowledge that Supports Teaching Based Development of Agents*, PhD Dissertation in Information Technology, George Mason University, Fairfax, Virginia, USA, 2002.
[28] Mitchell, T.M., Keller, T. and Kedar-Cabelli, S., Explanation-Based Generalization: A Unifying View, *Machine Learning,* Vol. 1, pp. 47-80, 1986.
[29] DeJong, G. and Mooney, R., Explanation-Based Learning: An Alternative View, *Machine Learning*, 1:145-176, 1986.
[30] Boicu C., Tecuci G., Boicu M., Improving Agent Learning through Rule Analysis, in *Proc. of the Int. Conf. on Artificial Intelligence*, ICAI-05, Las Vegas, June 27-30, 2005.
[31] Tecuci G., Boicu M., Hajduk T., Marcu D., Barbulescu M., Boicu C., Le V., An Approach to Rapid Development of Virtual Experts for Multi-Domain Collaborative Planning, *Research Report,* Learning Agents Center, August 2005.

Gheorghe Tecuci is Professor of Computer Science in the Volgenau School of Information Technology and Engineering and Director of the Learning Agents Center at George Mason University. He is also a member of the Romanian Academy and former Chair of Artificial Intelligence at the US Army War College. He received two Ph.D.s in Computer Science, from the University of Paris-South and from the Polytechnic University of Bucharest, and has published over 150 papers.

Mihai Boicu is Assistant Professor of Information Technology and Associate Director of the Learning Agents Center at George Mason University. He received a License in Informatics from the Bucharest University in 1995, and a Ph.D. in Information Technology from George Mason University in 2003. He has published around 50 papers.

Dorin Marcu is a Ph.D. candidate and a Research Assistant in the Learning Agents Center. He holds a B.S. in Computer Science from Polytechnic University of Bucharest, Romania, and has published over 30 conference and journal papers.

Marcel Barbulescu is a Ph.D. candidate and a Research Assistant in the Learning Agents Center. He received a B.S. in Computer Science from Polytechnic University of Timisoara, Romania, in 2001, and has published over 10 papers.

Cristina Boicu is a Research Assistant Professor in the Learning Agents Center at George Mason University since she received her Ph.D. in Computer Science from George Mason University, in Fall 2006. She has published over 20 papers.

Vu Le is a Research Instructor and a Ph.D. candidate in the Learning Agents Center. He received an M.S. in Computer Science, in 1999, from George Mason University and has published several papers.

Thomas Hajduk is former president of the Virginia Emergency Management Association, with over 30 years of experience as Fire Division Chief, Fire Marshal, or Deputy Coordinator of Emergency Services, in Prince William County, Virginia. He received a B.S. in Fire Administration and Technology from George Mason University, in 1981.