In Teodorescu, H.N., Mlynek, D., Kandel, A. and Zimmermann, H.J. (eds).
Intelligent Systems and Interfaces, *Kluwer Academic Press. 2000.*

Chapter 1

A Tutoring Based Approach to the Development of Intelligent Agents

Gheorghe Tecuci, Mihai Boicu, Kathryn Wright, Seok Won Lee, Dorin Marcu and Michael Bowman
Learning Agents Laboratory, Computer Science Department, George Mason University

Key words: intelligent agents, knowledge acquisition, machine learning, planning, knowledge representation

Abstract: This chapter introduces the concept of intelligent agent, analyses some of the issues and trends in developing them and presents a specific agent development approach. The presented approach, called Disciple, relies on importing ontologies from existing repositories of knowledge, and on teaching the agent how to perform various tasks, in a way that resembles how an expert would teach a human apprentice when solving problems in cooperation.

1. INTELLIGENT AGENTS

Significant advances in computer technology and in the various areas of Artificial Intelligence (such as knowledge representation, problem solving and planning, learning, natural language processing, and vision) have been made in the last decade. These advances make feasible the building of systems that exhibit not just one but several of the characteristics that we associate with intelligence in human behavior. Such systems, called intelligent agents, have the ability to perceive their environment, can reason to interpret perceptions, draw inferences, solve problems, and determine actions, and can act upon that environment to realize a set of goals or tasks for which they were designed. An intelligent agent interacts with a human or some other agents via some kind of agent-communication language. It may not obey commands blindly, but may have the ability to modify requests, ask

clarification questions, or even refuse to satisfy certain requests. The agent can accept high-level requests indicating what the user wants and can decide how to satisfy each request with some degree of independence or autonomy, exhibiting goal-directed behavior and dynamically choosing which actions to take, and in what sequence. It can collaborate with its user to improve the accomplishment of his/her tasks or can carry out such tasks on the user's behalf. In so doing, it employs some knowledge or representation of the user's goals or desires. It can monitor events or procedures for the user, can advise the user on how to perform a task, can train or teach the user, or can help different users collaborate.

The behavior of the agent is based on a correspondence between the external application domain of the agent and an internal model of this domain consisting of a knowledge base and an inference engine (see Figure 1). The knowledge base contains the data structures representing the entities from the agent's application domain such as objects, relations between objects, classes of objects, laws and actions. The inference engine consists of the programs that manipulate the data structures in the knowledge base in order to solve the problems for which the agent was designed.

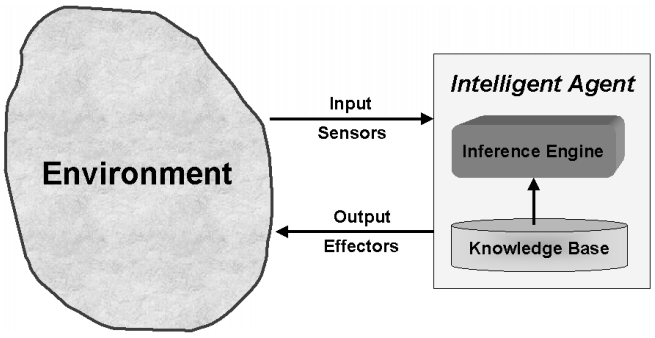

Figure 1. The overall architecture of an intelligent agent

2. GENERAL ISSUES AND TRENDS IN THE DEVELOPMENT OF INTELLIGENT AGENTS

Manual acquisition of knowledge from human experts by knowledge engineers is the most common approach to the process of developing an intelligent agent, and a knowledge-based system, in general. As illustrated in Figure 2, a knowledge engineer interacts with a domain expert to understand how the expert solves problems and what knowledge he or she uses. Then

the knowledge engineer chooses the representation of knowledge, builds the inference engine, elicits knowledge from the expert, conceptualizes it and represents it in the knowledge base. This knowledge elicitation and representation process is particularly difficult because the form in which the expert expresses his or her knowledge is significantly different from how it should be represented in the knowledge base. Moreover, the expert typically fails to specify the knowledge that is common sense or implicit in human communication, but which needs to be explicitly represented in the knowledge base. After the knowledge is elicited it has to be verified by the expert with the knowledge engineer making corrections in the knowledge base. This indirect transfer of knowledge, between the domain expert and the knowledge base, through the knowledge engineer, leads to a long, painful and inefficient knowledge base development process.

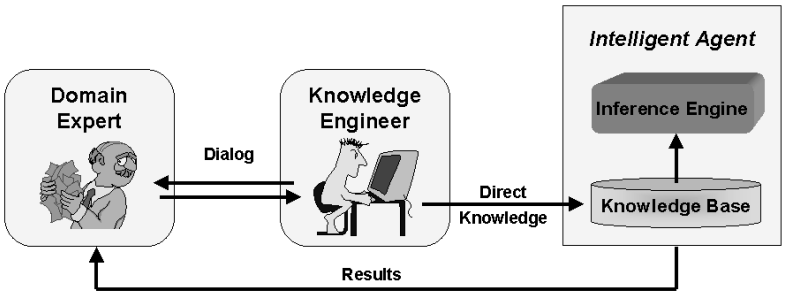

Figure 2. Overview of manual knowledge acquisition

Some of the issues that have been found to be limiting factors in developing intelligent agents for a wide range of problems and domains are:

– finding the right balance between using general tools and developing domain specific modules;
– limited ability to reuse previously developed knowledge;
– the knowledge acquisition bottleneck;
– the knowledge adaptation bottleneck;
– the scalability of the agent building process;
– the portability of the agent building tools and of the developed agents;
– slow development process.

We will briefly address these issues, as well as current research trends in dealing with them.

When developing an agent, it is important to find a suitable balance between reusing general modules and building specific modules. Reusing general modules significantly speeds up the development process. However, the agent may not be well adapted to its specific application domain and may not be that useful. On the contrary, building the agent from domain-specific

modules leads to a well-adapted and useful agent, but the development process is very difficult. Many of existing agent-building tools provide an inference engine, a representation formalism in which the knowledge base could be encoded, and mechanisms for acquiring, verifying or revising knowledge expressed in that formalism. These tools trade power (i.e., the assistance given to the expert) against generality (i.e., their domain of applicability), covering a large spectrum. At the power end of the spectrum there are tools customized to a problem-solving method and a particular domain (Musen and Tu, 1993). At the generality end are the tools applicable to a wide range of tasks or domains, such as CLIPS (Giarratano and Riley, 1994). In between are tools that are method-specific and domain independent (Chandrasekaran and Johnson, 1993).

All the existing agent building tools exploit, to a certain degree, the architectural separation between the general inference engine and the application-specific knowledge base, with the goal of reusing the inference engine for a new agent. Existing knowledge bases are very rarely reused, primarily for two reasons. First, the knowledge in the knowledge base is usually very specific to a particular domain and problem, and cannot be applied directly to a different application area. Second, even if the knowledge base of an agent is directly relevant to the new area, reuse of it by a different agent that uses a different knowledge representation, is likely to be very difficult because of the differences between the knowledge models of the two agents. This situation, however, is currently changing. First, we are witnessing a new architectural separation at the level of the knowledge base. The knowledge base is increasingly regarded as consisting of two main components: an ontology that defines the concepts of the application domain, and a set of problem solving rules expressed in terms of these concepts. While an ontology is characteristic to a certain domain (such as an ontology of military units, or an ontology of military equipment), the rules are much more specific, corresponding to a certain type of application in that domain. For example, there may be rules for an agent that assists a commander in critiquing courses of action, or rules for an agent that assists in planning the repair of damaged bridges or roads. This emergence of domain ontologies is primarily a result of terminological standardization to facilitate automatic processing of information, particularly information retrieval. Some examples of domain or general purpose ontologies are UMLS (UMLS 1998), CYC (Lenat 1995), and WordNet (Fellbaum 1998). The availability of domain ontologies raises the prospects of sharing and reusing them when building a new agent. Recently, the Open Knowledge Base Connectivity (OKBC) protocol has been defined to facilitate knowledge sharing and reuse (Chaudhri et al. 1998). OKBC is a standard for accessing knowledge bases stored in different frame representation systems.

It provides a set of operations for a generic interface to such systems. There is also an ongoing effort of developing OKBC servers for various systems, such as Ontolingua (Farquhar et al. 1996) and Loom (MacGregor 1991). These servers are becoming repositories of reusable ontologies and domain theories, and can be accessed using the OKBC protocol.

There are two very difficult problems in developing an intelligent agent: the encoding of knowledge in the knowledge base (known as "the knowledge acquisition bottleneck"), and the modification of this knowledge in response to changes in the application domain or in the requirements of the agent ("the knowledge maintenance bottleneck"). A promising approach to both of these problems is to develop a learning agent that is able to acquire and maintain its knowledge by itself. In addition to the knowledge base and the inference engine, the architecture of a learning agent includes a learning engine consisting of the programs that create and update the data structures in the knowledge base. The learning agent could learn from a variety of information sources in the environment. It may learn from its user or from other agents, either by being directly instructed by them or just by observing and imitating their behavior. It may learn from a repository of information (such as a data base) or it may learn from its own experience. Building a practical autonomous learning agent that can acquire and update its knowledge by itself is not yet practical; we do not yet understand enough about the cognitive process of learning. Therefore, a more practical approach is to develop an interactive learning agent that can interact with an expert. Such an agent can perform most of the functions of the knowledge engineer. It allows the expert to communicate expertise in a way familiar to him/her and is responsible for building, updating and reorganizing the knowledge base.

Obviously, the usefulness and generality of the intelligent agents and of the agent building tools are significantly enhanced if they are portable. Therefore, a current trend in developing them is the use of Common Lisp (for the core functionality) and of JAVA (for the interface).

Finally, as the history of Artificial Intelligence makes clear, the fact that one approach worked in the development of a small scale agent is in no way a guarantee that it will also work for building agents for complex, real-world applications. Therefore, the scalability of the agent building process is an important aspect of any agent building methodology and tool.

3. THE DISCIPLE APPROACH FOR DEVELOPING INTELLIGENT AGENTS AND AN EXEMPLARY AGENT

Disciple is an apprenticeship, multistrategy learning approach for developing intelligent agents, which addresses the design issues discussed in the previous section. In the Disciple approach, an expert teaches the agent how to perform domain-specific tasks in a way that resembles how the expert would teach an apprentice, by giving the agent examples and explanations as well as by supervising and correcting its behavior (Tecuci, 1998; Tecuci et al., 1999). This approach integrates many machine learning and knowledge acquisition techniques (such as inductive learning from examples, explanation-based learning, learning by analogy, learning by experimentation) taking advantage of their complementary strengths to compensate for their weaknesses (Michalski and Tecuci, 1994; Tecuci and Kodratoff, 1995). As a consequence, the Disciple approach significantly reduces the involvement of the knowledge engineer in the process of building an intelligent agent. The current version of the Disciple approach is implemented in the Disciple Learning Agent Shell (Disciple-LAS). A learning agent shell consists of a learning and knowledge acquisition engine as well as an inference engine and supports building an agent with a knowledge base consisting of an ontology and a set of problem solving rules.

With respect to the Disciple-LAS shell and methodology we have formulated the following three claims:

- they enable rapid acquisition of relevant problem solving knowledge from subject matter experts, with limited assistance from knowledge engineers;
- the acquired problem solving knowledge is of a good enough quality to assure a high degree of correctness of the solutions generated by the agent;
- the acquired problem solving knowledge assures a high performance of the problem solver.

In the rest of this chapter we will present the Disciple agent building approach using, as an example, the process of building an agent for solving the workaround challenge problem. We will first define the workaround challenge problem. Then we will introduce Disciple based modeling of an application domain. Next we will present the architecture of the Disciple-LAS and the agent building methodology. Finally, we will present experimental results from building the specified agent and summarize our conclusions.

This workaround problem consists of assessing how rapidly and by what method a military unit can reconstitute or bypass damage to an

infrastructure, such as a damaged bridge (Alphatech 1998; Cohen et al. 1998).

The input to the agent includes three elements:

– a description of the military unit that needs to work around some damage (e.g. an armored tank brigade or a supply company),
– a description of the damage (e.g. a span of the bridge is dropped and the area is mined), and of the terrain (e.g. the soil type, the slopes of the river banks, the river's speed, depth and width),
– a detailed description of the resources in the area that could be used to repair the damage. This includes a description of the engineering assets of the military unit that has to work around the damage, as well as the descriptions of other military units in the area that could provide additional resources.

The output of the agent consists of the most likely repair strategies, each described in terms of three elements:

– a reconstitution schedule, giving the transportation capacity of the damaged link (bridge, road or tunnel), as a function of time, including both a minimum time and an expected time;
– a time line of engineering actions to perform the repair, the minimum as well as the expected time that these actions require, and the temporal constraints among them; and
– a set of required assets for the entire strategy and for each action.

Workaround generation requires detailed knowledge about the capabilities of engineering equipment and its use. For example, repairing damage to a bridge typically involves different types of mobile bridging equipment and earth moving equipment. Each kind of mobile bridge takes a characteristic amount of time to deploy, requires different kinds of bank preparation, and is owned by different echelons in the military hierarchy. This information was available from military experts and Army field manuals.

4. DOMAIN MODELING FOR INTEGRATED KNOWLEDGE REPRESENTATION, KNOWLEDGE ACQUISITION, LEARNING AND PROBLEM SOLVING

The Disciple modeling of an application domain provides a natural way to integrate knowledge representation, knowledge acquisition, learning and problem solving, into an end-to-end shell for building practical, knowledge-based agents. We have adopted the classical task reduction paradigm as the problem solving approach. In this paradigm, a task to be accomplished by

the agent is successively reduced to simpler tasks, until the initial task is reduced to a set of elementary tasks that can be immediately performed. Within this paradigm, an application domain is modeled based on six types of knowledge elements: objects, features, tasks, examples, explanations, and problem reduction rules.

The ***objects*** represent either specific individuals or sets of individuals (also called concepts) in the application domain. The objects are hierarchically organized according to the generalization relation. Figure 3, for instance, presents a partial hierarchy of objects of the workaround agent. Included are several types of military bridges that can be used to cross a river.

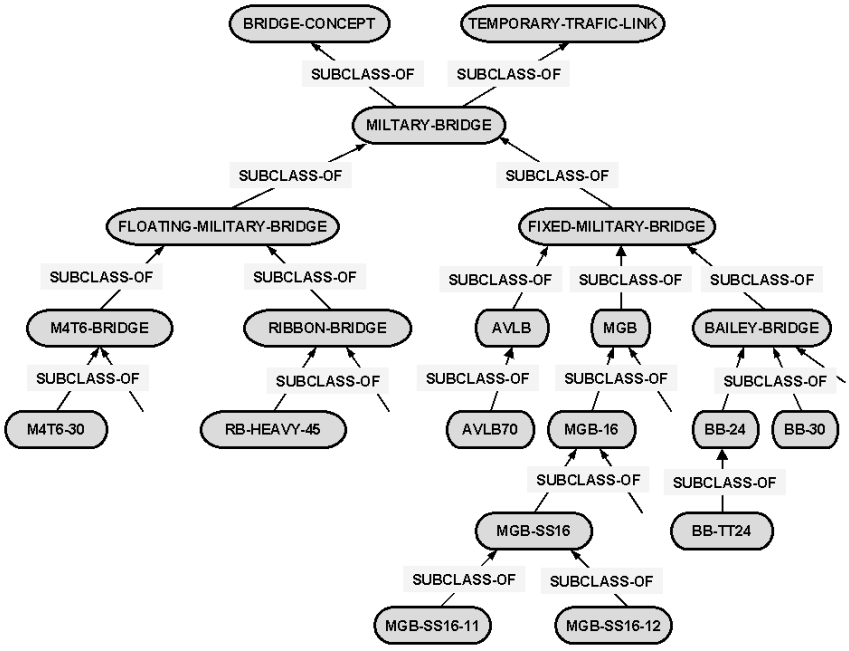

Figure 3. A sample of the object hierarchy

The ***features*** and the sets of features are used to further describe objects, other features and tasks. For instance, Figure 4 contains the descriptions of two objects from the hierarchy in Figure 3, AVLB (an acronym for armored vehicle launched bridge) and AVLB70. An AVLB is a type of fixed-military bridge that has additional features. AVLB70 is a type of AVLB bridge. Each such object (concept) inherits all of the features of its superconcepts. Therefore, all the features of AVLB are also features of AVLB70.

AVLB	SUBCLASS-OF	FIXED-MILITARY-BRIDGE
	MIN-CROSSING-TIME-FOR-UNSTABILIZED-END	2 MIN
	EXPECTED-CROSSING-TIME-FOR-UNSTABILIZED-END	10 MIN
	MIN-EMPLACEMENT-TIME	5 MIN
	EXPECTED-EMPLACEMENT-TIME	10 MIN
	MAX-DOWNHILL-SLOPE-FOR-EQ	19 %
	MAX-TRANSVERSE-SLOPE	11 %
	MAX-UPHILL-SLOPE-FOR-EQ	28 %
AVLB70	SUBCLASS-OF	AVLB
	HAS-WIDTH	19.2 METERS
	MAX-GAP	17 METERS
	MAX-REDUCIBLE-GAP	26 METERS
	MLC-RATING	70 TONS
	WEIGHT	15 TONS

Figure 4. Descriptions of two objects from the hierarchy in Figure 3

The features are defined in the same way as the objects, in terms of more general features. Figure 5, for instance, presents a sample of the feature hierarchy. Two important characteristics of any feature are its domain (the set of objects that could have this feature) and its range (the set of possible values of the feature). The features may also specify functions for computing their values.

A ***task*** is a representation of anything that the agent may be asked to accomplish. The following, for instance, is the description of the task to workaround an unmined destroyed bridge by using a fixed military bridge:

WORKAROUND-UNMINED-DESTROYED-BRIDGE-WITH-FIXED-BRIDGE	
AT-LOCATION	SITE100
FOR-GAP	SITE103
BY-UNIT	UNIT91010

The bridge is at location SITE100, the river gap crossed by the bridge is SITE103, and the military unit to perform the workaround is UNIT91010. Each of these objects is an element of the object hierarchy, and is described by its own features and values. Similarly, the features AT-LOCATION, FOR-GAP, and BY-UNIT, are elements of the feature hierarchy. The task itself is an element of the task hierarchy.

The objects, features and tasks are represented as frames, according to the OKBC knowledge model, with some extensions (Chaudhri et al. 1998).

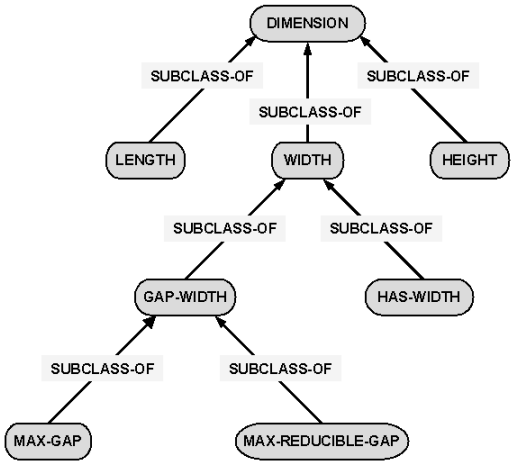

Figure 5. A sample of the feature hierarchy

The ***examples*** represent specific task reductions, and have the following general form:

TR: IF the task to accomplish is T_1
THEN accomplish the tasks T_{11}, … , T_{1n}

A task may be reduced to one simpler task, or to a (partially ordered) set of tasks. Correct task reductions are called positive examples and incorrect ones are called negative examples. An example of task reduction is presented in Figure 6. It states that in order to work around the damaged bridge at SITE100, one has to use a bridge equipment of type AVLB-EQ and to reduce the size of the gap.

An ***explanation*** of a task reduction is an expression of objects and features that indicates why a task reduction is correct (or why it is incorrect). It corresponds to the justification given by a domain expert for a specific task reduction:

the task reduction TR is correct because E

One could more formally represent the relationship between a task reduction TR and its explanation E as follows:

E ➔ TR, or E ➔ (accomplish(T_1) ➔ accomplish(T_{11}, ... , T_{1n}))

This interpretation is useful in a knowledge acquisition and learning context where the agent tries to learn how to accomplish a task and why the solution is correct from a domain expert.

IF the task to accomplish is

WORKAROUND-UNMINED-DESTROYED-BRIDGE-WITH-FIXED-BRIDGE

AT-LOCATION	SITE100
FOR-GAP	SITE103
BY-UNIT	UNIT91010

THEN accomplish the task

USE-FIXED-BRIDGE-WITH-GAP-REDUCTION-OVER-GAP

AT-LOCATION	SITE100
FOR-GAP	SITE103
BY-UNIT	UNIT91010
WITH-BR-EQ	AVLB-EQ

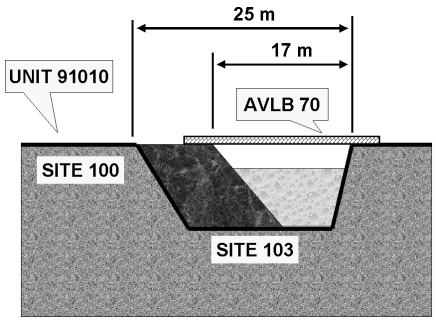

Figure 6. An example of task reduction

For example, an explanation of the task reduction from Figure 6 is the one from Figure 7.

SITE103	HAS-WIDTH	25M,		
AVLB-EQ	CAN-BUILD	AVLB70	MAX-GAP	17M < 25M
SITE103	HAS-WIDTH	25M,		
AVLB-EQ	CAN-BUILD	AVLB70	MAX-REDUCIBLE-GAP	26M = 25M
UNIT91010	MAX-WHEELED-MLC	20T,		
AVLB-EQ	CAN-BUILD	AVLB70	MLC-RATING	70T = 20T
UNIT91010	MAX-TRACKED-MLC	40T,		
AVLB-EQ	CAN-BUILD	AVLB70	MLC-RATING	70T = 40T

Figure 7. Explanations of the task reduction in Figure 6

The first two explanation pieces justify why one needs to use gap reduction. The width of the SITE103 gap is 25 m and the AVLB-EQ allows building a bridge of type AVLB70 that can only span gaps up to 17 m. Therefore, the gap is too wide to install AVLB70 directly. However, any gap that is smaller than 26 m can be reduced to a 17 m gap on which one can

install an AVLB70 bridge. The next two explanation pieces show that an AVLB70 bridge is strong enough to sustain the vehicles of UNIT91010. Indeed, the maximum load class of the wheeled vehicles of UNIT91010 is 20 tons and AVLB70 can sustain vehicles with a load of up to 70 tons. Similarly, the AVLB70 bridge can sustain the tracked vehicles of UNIT91010.

The relationship between the task reduction TR and its explanation E can also be represented in the equivalent form:

$((\text{accomplish}(T_1) \,\&\, E) \rightarrow \text{accomplish}(T_{11}, \ldots, T_{1n}))$

which, in a problem solving context, is interpreted as:

IF the task to accomplish is T_1 and
E holds
THEN accomplish the tasks $T_{11}, \ldots, T_{1n}$

Finally, the ***task reduction rules*** represent general reductions of tasks to subtasks as well as the conditions when such reductions can be performed:

IF the task to accomplish is T_{1g} and
E_h holds
THEN accomplish the tasks $T_{11g}, \ldots, T_{1ng}$

In addition to the rule's condition that needs to hold in order for the rule to be applicable, the rule may also have several EXCEPT-WHEN conditions that should not hold, in order for the rule to be applicable. The rule may also have EXCEPT-FOR conditions (that specify negative exceptions of the rule) and FOR conditions (that specify positive exceptions).

In Disciple, the task reduction rules are learned by the agent through an interaction with the domain expert. The ontology of objects, features and tasks serves as the generalization hierarchy for Disciple. An example is basically generalized by replacing its objects with more general objects from the ontology. Another important aspect of Disciple is that the ontology is itself evolving during knowledge acquisition and learning. This distinguishes Disciple from most of the other learning agents that make the less realistic assumption that the representation language for learning is completely defined before any learning can take place.

Because the Disciple agent is an incremental learner, most often its rules are only partially learned. A partially learned rule has two conditions, a plausible upper bound (PUB) condition E_g which, as an approximation, is more general than the exact condition E_h, and a plausible lower bound (PLB) condition E_s which, as an approximation, is less general than E_h:

IF the task to accomplish is T_{1g} and
PUB: E_g holds
PLB: E_s holds
THEN accomplish the tasks $T_{11g}, \ldots, T_{1ng}$

We will refer to such a rule as a plausible version space rule, or PVS rule. Plausible version space rules are used in problem solving to generate task reductions with different degrees of plausibility, depending on which of its conditions are satisfied. If the PLB condition is satisfied, then the reduction is very likely to be correct. If PLB is not satisfied, but PUB is satisfied, then the solution is considered only plausible. The same rule could also be applied for tasks that are considered similar to T_{1g}. In such a case the reductions are considered even less plausible. Any application of a PVS rule however, either successful or not, provides an additional (positive or negative) example, and possibly an additional explanation piece, that are used by the agent to further improve the rule. The plausible version space rule learned from the example in Figure 6 is presented in Figure 8.

IF *the task to accomplish is*
WORKAROUND-UNMINED-DESTROYED-BRIDGE-WITH-FIXED-BRIDGE
AT-LOCATION ?O1
FOR-GAP ?O2
BY-UNIT ?O3

Plausible upper bound			*Plausible lower bound*		
?O1	IS	BRIDGE	?O1	IS	SITE100
?O2	IS	CROSS-SECTION	?O2	IS	SITE103
	HAS-WIDTH	?N4		HAS-WIDTH	?N4
?O3	IS	MILITARY-UNIT	?O3	IS	UNIT91010
	MAX-TRACKED-MLC	?N3		MAX-TRACKED-MLC	?N3
	MAX-WHEELED-MLC	?N2		MAX-WHEELED-MLC	?N2
?O4	IS	AVLB-EQ	?O4	IS	AVLB-EQ
	CAN-BUILD	?O5		CAN-BUILD	?O5
	MAX-REDUCIBLE-GAP	?N5		MAX-REDUCIBLE-GAP	?N5
	MAX-GAP	?N6		MAX-GAP	?N6
?O5	IS	AVLB70	?O5	IS	AVLB70
	MLC-RATING	?N1		MLC-RATING	?N1
?N1	IS-IN	[0.0 150.0]	?N1	IS-IN	[70.0 70.0]
?N2	IS-IN	[0.0 150.0]	?N2	IS-IN	[25.0 25.0]
	≤	?N1		≤	?N1
?N3	IS-IN	[0.0 150.0]	?N3	IS-IN	[63.0 63.0]
	≤	?N1		≤	?N1
?N4	IS-IN	[0.0 100.0]	?N4	IS-IN	[25.0 25.0]
?N5	IS-IN	[0.0 100.0]	?N5	IS-IN	[26.0 26.0]
	≥	?N4		≥	?N4
?N6	IS-IN	[0.0 100.0)	?N6	IS-IN	[17.0 17.0]
	<	?N4		<	?N4

THEN accomplish the task
USE-FIXED-BRIGE-WITH-GAP-REDUCTION-OVER-GAP
AT-LOCATION ?O1
FOR-GAP ?O2
BY-UNIT ?O3
WITH-BR-EQ ?O4

Figure 8. A task reduction rule

The workaround agent receives as input the description of damage and generates a plan to work around it. Figure 9 presents an example of such a plan.

Workaround summary

Initial task:
WORKAROUND-DAMAGE
FOR-DAMAGE DAMAGE200
BY-INTERDICTED-UNIT UNIT91010

Engineering action: INSTALL AVLB
MIN-DURATION 11H:4M:58S
EXPECTED-DURATION 14H:25M:56S
RESOURCES REQUIRED (AVLB-UNIT202 BULLDOZER-UNIT201)
LINK CAPACITY AFTER RECONSTRUCTION 135.13 VEHICLES/HR

Detailed plan:

S1 OBTAIN-OPERATIONAL-CONTROL-FROM-CORPS
OF-UNIT UNIT202
BY-UNIT UNIT91010
MIN-DURATION 4H:0M:0S
EXPECTED-DURATION 6H:0M:0S
TIME-CONSTRAINTS: NONE

S2 MOVE-UNIT
FOR-UNIT UNIT202
FROM-LOCATION SITE0
TO-LOCATION SITE100
MIN-DURATION 1H:8M:14S
EXPECTED-DURATION 1H:8M:14S
TIME-CONSTRAINTS: AFTER S1

S3 REPORT-OBTAINED-EQUIPMENT
FOR-EQ-SET AVLB-UNIT202
MIN-DURATION 0S
EXPECTED-DURATION 0S
TIME-CONSTRAINTS: AFTER S2

S4 OBTAIN-OPERATIONAL-CONTROL-FROM-CORPS
OF-UNIT UNIT201
BY-UNIT UNIT91010
MIN-DURATION 4H:0M:0S
EXPECTED-DURATION 6H:0M:0S
TIME-CONSTRAINTS: NONE

S5 MOVE-UNIT
FOR-UNIT UNIT201
FROM-LOCATION SITE0
TO-LOCATION SITE100
MIN-DURATION 1H:8M:14S
EXPECTED-DURATION 1H:8M:14S
TIME-CONSTRAINTS: AFTER S4

S6 REPORT-OBTAINED-EQUIPMENT
FOR-EQ-SET BULLDOZER-UNIT201
MIN-DURATION 0S
EXPECTED-DURATION 0S
TIME-CONSTRAINTS: AFTER S5

S7 NARROW-GAP-BY-FILLING-WITH-BANK
FOR-GAP SITE103
FOR-BR-DESIGN AVLB70
MIN-DURATION 5H:19M:44S
EXPECTED-DURATION 6H:7M:42S
RESOURCES-REQUIRED BULLDOZER-UNIT201
TIME-CONSTRAINTS: AFTER S6

S8 EMPLACE-AVLB
FOR-BR-DESIGN AVLB70
MIN-DURATION 5M:0S
EXPECTED-DURATION 10M:0S
RESOURCES-REQUIRED AVLB-UNIT202
TIME-CONSTRAINTS: AFTER S3, S7

S9 REPORT-EMPLACED-FIXED-BRIDGE
FOR-MIL-BRIDGE IXED-MILITARY-BRIDGE-EQ
MIN-DURATION 0S
EXPECTED-DURATION 0S
TIME-CONSTRAINTS: AFTER S8

S10 MOVE-EQUIPMENT-OVER-UNSTABILIZED-MIL-BRIDGE
FOR-EQ-SET BULLDOZER-UNIT201
FOR-BR-DESIGN AVLB70
MIN-DURATION 2M:0S
EXPECTED-DURATION 10M:0S
RESOURCES-REQUIRED AVLB-UNIT202
TIME-CONSTRAINTS: AFTER S9

S11 MINOR-BANK-PREPARATION
OF-BANK SITE105
MIN-DURATION 30M:0S
EXPECTED-DURATION 50M:0S
RESOURCES-REQUIRED BULLDOZER-UNIT201
TIME-CONSTRAINTS: AFTER S10

S12 RESTORE-TRAFFIC-LINK
FOR-UNIT UNIT91010
FOR-LINK AVLB70
LINK-CAPACITY 135.13 VEHICLES/H
MIN-DURATION 0S
EXPECTED-DURATION 0S
TIME-CONSTRAINTS: AFTER S11

Figure 9. A generated workaround plan

The damage is a destroyed bridge and UNIT91010 has to work around it. The best plan consists of installing an AVLB bridge over the river gap. It is estimated that this will take a minimum of 11h:4m:58s, the expected duration being 14h:25m:56s. UNIT91010 will need the help of UNIT202, which has AVLB equipment, and of UNIT201, which has a bulldozer. After the bridge is installed, it will allow a traffic rate of 135.13 vehicles/h. The plan consists of 12 elementary actions. UNIT91010 has to obtain operational control of UNIT202 which has the AVLB. Then this unit has to come to the site of the destroyed bridge. Also, UNIT91010 has to obtain operational control of UNIT201 which has a bulldozer. This unit will have to move to the site of the destroyed bridge and then to narrow the river gap from 25m to 17m. These

actions can take place in parallel with the actions of bringing UNIT202 to the bridge site. Then the AVLB bridge is emplaced, the bulldozer moves over the bridge and clears the other side of the river to restore the flow of traffic. This plan was generated by successively reducing the WORKAROUND-DAMAGE task to simpler subtasks, until this task was reduced to the 12 tasks in Figure 9.

5. ARCHITECTURE OF THE DISCIPLE LEARNING AGENT SHELL

The architecture of Disciple-LAS is presented in Figure 10. It includes seven main components, shown in the light gray area, which are domain independent:

– a knowledge acquisition and learning component for developing and improving the KB. It contains several modules for rule learning, rule refinement, and exception handling, and a set of browsers and editors, each specialized for one type of knowledge (objects, features, tasks, examples, explanations and rules).
– a domain-independent problem solving engine based on task reduction. It supports both interactive (step by step) problem solving and autonomous problem solving.
– a knowledge import/export component for accessing remote ontologies located on OKBC servers.
– a knowledge base manager which controls access and updates to the knowledge base. Each module of Disciple can access the knowledge base only through the functions of the KB manager.
– an OKBC layer which assures a uniform management of all the elements of the knowledge base, according to the OKBC knowledge model. It also allows future integration with Disciple of efficient memory management systems, such as PARKA (Stoffel et al. 1997).
– an initial domain-independent knowledge base to be developed for the specific application domain. This knowledge base contains the elements that will be part of each knowledge base built with Disciple, such as an upper-level ontology.
– a window-based, domain-independent, graphical user interface, intended to be used primarily by the knowledge engineer.

The two components in the dark gray area are the domain dependent components that need to be developed and integrated with the Disciple-LAS shell to form a customized agent for a specific application. They are:

– a domain-dependent graphical user interface which is built for the specific agent to allow the domain experts to communicate with the agent

in a manner as close as possible to the way they communicate in their environment.

– a domain-specific problem solving component that extends the basic task-reduction engine in order to satisfy the specific problem solving requirements of the application domain.

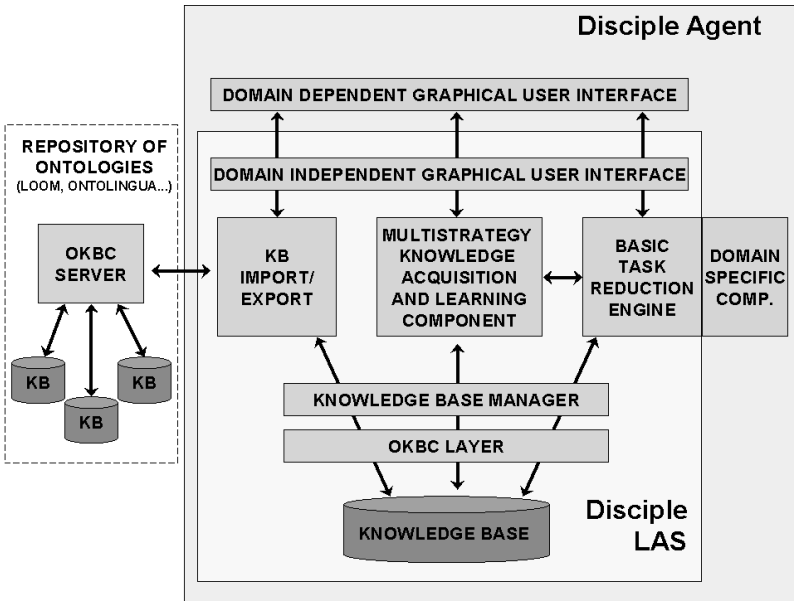

Figure 10. General architecture of Disciple-LAS

In order to assure the portability of the shell, the interface is implemented in JAVA and all the other components are implemented in Common LISP.

6. THE METHODOLOGY OF BUILDING DISCIPLE AGENTS

In this section we will briefly present the main steps of the integrated Disciple-LAS methodology for building end-to-end agents. We will stress the characteristic features of this methodology and we will illustrate them with informal intuitive examples from the development of the workaround agent described above. The steps of the methodology are to be executed in sequence but at each step one can return to any of the previous steps to fix any discovered problem.

6.1 Specification of the problem

The domain expert and the knowledge engineer generally accomplish this step together. The workaround challenge problem was defined in a 161-page report created by Alphatech (1998). This report also identified many of the concepts which needed to be represented in the agent's ontology, such as military units, engineering equipment, types of damage, and geographical features of interest. Therefore, it provided a significant input to the ontology building process.

6.2 Modelling the problem solving process as task reduction

Once the problem is specified, the expert and the knowledge engineer have to model the problem solving process as a task reduction, because this is the problem solving approach currently supported by the Disciple shell. However, the knowledge acquisition and learning methods of Disciple are general; one of our future research directions is to apply them in conjunction with other types of problem solvers. In the case of the workaround challenge problem, task reduction proved to actually be a very natural way to model the problem solving process because of the problem solver being a hierarchical non-linear planner. During the modeling process several informal task reduction trees were built. The top part of such a tree is presented in Figure 11. It represents a strategy to solve a certain class of workaround problems. There are several important results of the modeling process. The first is the conceptual task reductions that will guide the training of the agent by the domain expert. Informal descriptions of the agent's tasks are also produced. Additional necessary concepts and features are identified. Finally, the modeling process reveals similarities between different tasks.

6.3 Developing the customized agent

For the workaround domain, the task reduction engine had to be customized by including a component for ordering the generated plans based on the minimum time needed to execute them, and by generating a summary description of each plan. Additionally, an interface for displaying maps with the damaged area was integrated into the agent architecture.

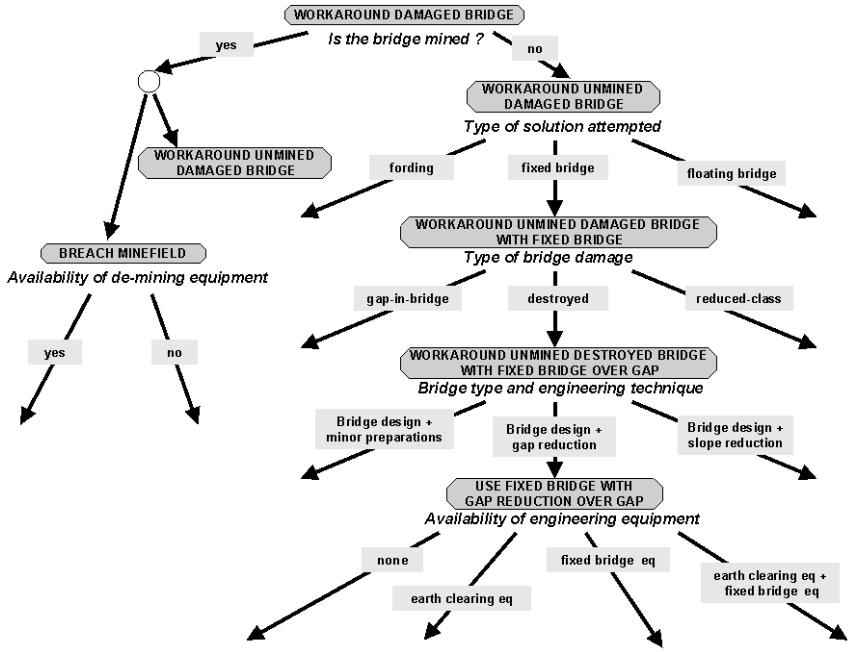

Figure 11. Informal task reduction tree

6.4 Importing concepts and features from other ontologies

As a result of the first two stages of the methodology, a significant number of necessary concepts and features will have been identified. Interacting with the Knowledge Base Import/Export Module, the domain expert attempts to import the descriptions of these concepts from an OKBC server. The expert can select a concept or its entire sub-hierarchy and the knowledge import module will introduce this new knowledge into Disciple's knowledge base.

In the case of the HPKB experiment, we imported from the LOOM server (MacGregor, 1991) elements of the military unit ontology, as well as various characteristics of military equipment (such as their tracked and wheeled vehicle military load classes). The extent of knowledge import was more limited than it could have been because the LOOM ontology was developed at the same time as that of Disciple; we had to define concepts that were later defined in LOOM and could have been imported. In any case, importing those concepts proved to be very helpful, and has demonstrated the ability to reuse previously developed knowledge.

6.5 Extending the ontology

The Disciple shell contains specialized browsing and editing tools for each type of knowledge element. It contains an object editor, a feature editor, a task editor, an example editor, a rule editor, a hierarchy browser and an association browser. We have defined a specialized editor for each type of knowledge element to facilitate the interaction with the domain expert. Using these tools, the domain expert and the knowledge engineer will define the rest of the concepts and features identified in steps 1 and 2 (that could not be imported), as well as some or all of the tasks informally specified in step 3. The rest of the tasks, as well as new tasks, objects and features, can also be defined during the next step of training the agent.

6.6 Training the agent for its domain-specific tasks

During this step, the expert teaches Disciple to solve problems in a cooperative, step by step, problem solving session. The expert selects or defines an initial task and asks the agent to reduce it. The agent will try different methods to reduce the current task. First it will try to apply the rules with their exact or plausible lower bound conditions, because these are most likely to produce correct results. If no reduction is found, then it will try to use the rules considering their plausible upper bound conditions.

If the agent is not able to reduce the current task, then the solution must be defined by the expert. In such a case it represents an initial example for learning a new task reduction rule. To learn the rule, the agent will use the multistrategy learning method represented in Figure 12.

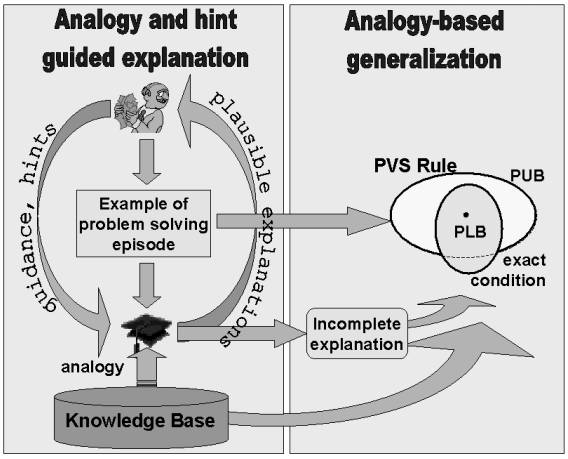

Figure 12. The rule learning method of Disciple

As Explanation-based Learning (DeJong and Mooney, 1986; Mitchell, Keller, Kedar-Cabelli, 1986), the learning method consists of two phases, explanation and generalization. However, in the explanation phase the agent is not building a proof tree, but only a justification. Also, the generalization is not a deductive one, but an analogy-based one.

Let us consider that the example provided by the expert is the one from Figure 6. The agent will first try to find an explanation of why the reduction is correct. This explanation is presented in Figure 7. Then the example and the explanation are generalized to the plausible version space rule from Figure 8. The most difficult part of this method is finding the explanation. The agent will attempt various heuristic strategies to propose plausible explanations from which the user will choose the correct ones. For instance, the agent will consider the rules that reduce the same task into different subtasks, and will use the explanations corresponding to these rules to propose similar explanations for the current reduction. This heuristic is based on the observation that the explanations of the alternative reductions of a task tend to have similar structures. The same factors are considered, but the relationships between them are different. Figure 13, for instance, shows three different reductions of the task to workaround a destroyed bridge.

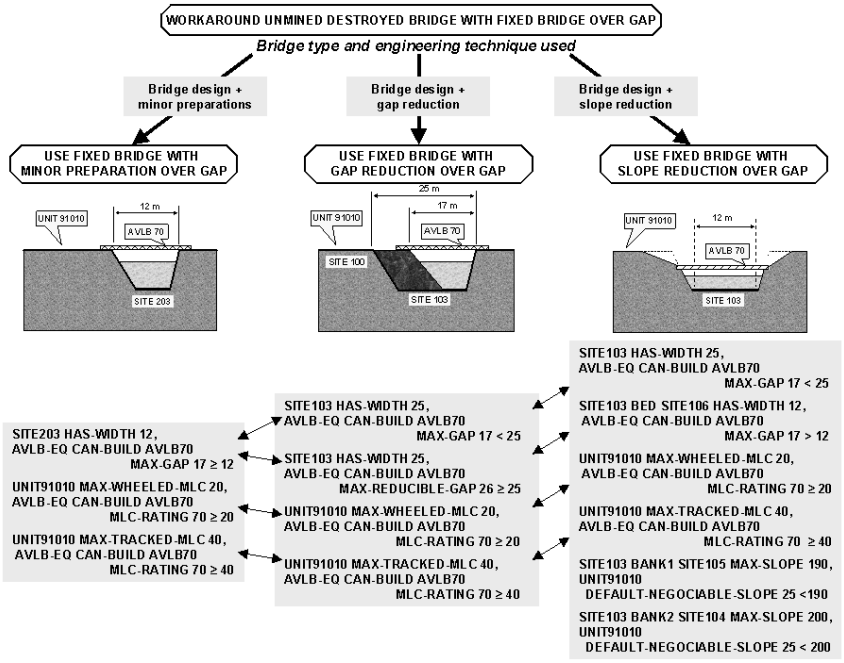

Figure 13. Generation of explanations by analogy with other reductions of the same task

The leftmost reduction consists of installing the fixed bridge with minor preparations, the center reduction consists of using gap reduction, and the rightmost reduction consists of using slope reduction. In a particular situation, the decision of which of these reductions to perform depends upon the specific relationships between the dimensions of the bridge and the dimensions of the river gap. Below each reduction in Figure 13 there are the explanations corresponding to it. Bi-directional arrows connect the similar explanations. As one can see, if the agent has already learned a rule corresponding to any of these reductions, then learning the rules corresponding to the other reductions is much simpler because the agent can propose explanations by analogy with those of the learned rule.

Another heuristic is to generate explanations of a task reduction by analogy with the explanations corresponding to the reduction of a similar task, as illustrated in Figure 14.

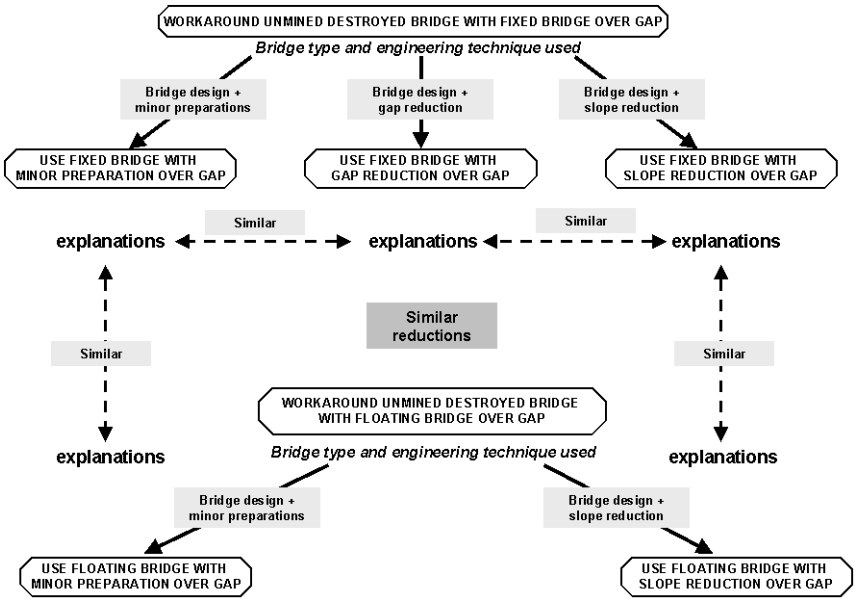

Figure 14. Generation of explanations by analogy with reductions of similar tasks

The bottom part of Figure 14 shows two possible reductions of the task of working around a destroyed bridge by using a floating bridge. The left hand side reduction consists of using the bridge with minor preparations, while the right hand side reduction consists in using the bridge with slope reduction. As indicated in Figure 14, the explanation of each of these task reductions is similar (and therefore could be generated by analogy) with the

explanations of the reductions from the top part of Figure 14 that consists in using a fixed military bridge. The goal of using these heuristics is to have the agent propose explanations ordered by their plausibility with the expert choosing the right ones, rather than requiring the expert to define them.

The above strategy works well when the agent has already learned rules similar to the rule currently being learned. In the situations when this is not true the agent has to acquire the explanations from the expert. However, even in such cases, the expert need not provide explanations, but only hints that may have various degrees of detail. Let us consider, for instance, the task reduction in Figure 6. The expert can give the agent a very general hint, such as, "Look for correlations between SITE103 (the river gap) and AVLB-EQ." A more specific hint would be "Look for correlations between the width of SITE103 and AVLB-EQ". An even more specific hint would be "Look for correlations between the width of SITE103 and the widths of the gaps breachable with AVLB-EQ." Such hints will guide the agent in looking for explanations that have a certain pattern, as indicated in the left hand side of Figure 15.

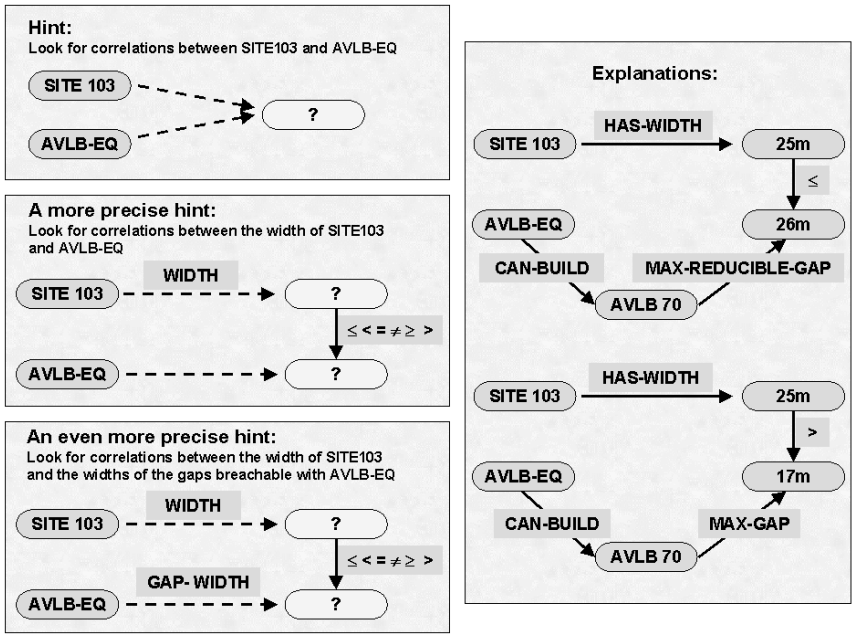

Figure 15. Guiding the agent to generate explanations

Among the plausible explanations proposed by the agent will be the correct explanations shown in the right hand side of Figure 15. Notice that

when matching the pattern corresponding to a hint with the explanation in the agent's ontology, the agent uses the generalization hierarchy of the features. For instance, WIDTH in the hint matches with HAS-WIDTH in the explanations because HAS-WIDTH is a subclass of WIDTH (see Figure 5). Also, GAP-WIDTH will match with both MAX-REDUCIBLE-GAP and MAX-GAP. The goal of this process is to allow the expert to provide hints or incomplete explanations rather than detailed explanations.

Let us now consider some of the other possible cases, where the agent proposes reductions based on the existing rules. If the expert accepted the reduction and it was obtained by applying the plausible upper bound condition of a rule, then the plausible lower bound condition of the rule is generalized to cover this reduction. If the reduction is rejected by the expert, then the agent will attempt to find an explanation of why the reduction is not correct (as described above), and will use this explanation to augment the applied rule with EXCEPT-WHEN conditions. When no such failure explanation is found, the agent may simply specialize the rule, to uncover the negative example. When this is not possible, the rule will be augmented with an EXCEPT-FOR condition. In a given situation, the agent may propose more than one solution. Each may be characterized separately as good or bad, and treated accordingly. Learning may be also postponed for some of these examples.

This training scenario encourages and facilitates knowledge reuse between different parts of the problem space, as has been experienced in the workaround domain. For instance, many of the rules corresponding to the AVLB bridges have been either generalized to apply to the bridges of types MGB and Bailey, or have been analogically used to generate examples and explanations to learn new rules for MGB and Bailey. These rules have been further generalized to apply to the floating bridges, or have been analogically used to generate examples and explanations to learn new rules for floating bridges. Then the rules for floating bridges have been used in a similar way for ribbon rafts. The rules for installing fixed bridges with slope reduction have been used similarly for fording with slope reduction, for ribbon bridges with slope reduction, and for ribbon rafts with slope reduction. The rules for installing fixed bridges over river gaps (with minor preparation or with gap reduction) have been generalized to rules for installing fixed bridges over gaps to also apply to craters. Many of the rules for de-mining craters applied also to demining bridges. The rules for reasoning about obtaining operational control of the engineering units and those for reasoning about position were generalized to apply to various types of workaround problems.

6.7 Testing and using the agent

During this phase the agent is tested with additional problems, the problem solver being used in autonomous mode to provide complete solutions without the expert's interaction. If any solution is not the expected one, then the expert enters the interactive mode to identify and help the agent fix the error, as described before. A non-expert user can use the developed agent. However, more interesting is the case where the agent continues to act as an assistant to the expert, solving problems in cooperation with and continuously learning from the expert, and becoming more and more useful.

In the case of the workaround domain, Alphatech provided a set of 20 testing problems, each with up to 9 different types of relevant solutions. These examples were used to train and test the agent.

6.8 Experimental Evaluation

The Disciple methodology and workaround agent were tested together with three other systems in a two week intensive study, in June 1998, as part of DARPA's annual HPKB program evaluation. The evaluation consisted of two phases, each consisting of a test and a re-test. In the first phase, the systems were tested on 20 problems provided by Alphatech that were similar with those used for systems development. Then the solutions were provided and the developers had one week to improve their systems, which were tested again on the same problems. In the second phase, the systems were tested on five new problems, partially or completely out of the scope of the systems. For instance, they specified a new type of damage (cratered roads), or required the use of new types of engineering equipment (TMM bridges, ribbon rafts and M4T6 rafts). Then again the correct solutions were provided and the developers had one week to improve and develop their systems, which were tested again on the same five problems and five new ones. Solutions were scored along five equally weighted dimensions:

1. generation of workaround solutions for all the viable options,
2. correctness of the overall time estimate for each workaround solution,
3. correctness of each solution step,
4. correctness of temporal constraints among these steps, and
5. appropriateness of engineering resources used.

Scores were assigned by comparing the systems' answers with those of Alphatech's expert. Bonus points were awarded when systems gave better answers than the expert did, and these answers were used as standard solutions for the next phase of the evaluation.

The participating teams were not uniform in terms of prior system development and human resources. Consequently, only one of them

managed to enter the evaluation with a fully developed agent, the other three having agents with incompletely developed knowledge bases. Figure 16 shows a plot of the overall coverage against the overall correctness of each system, for each of the two phases of the evaluation.

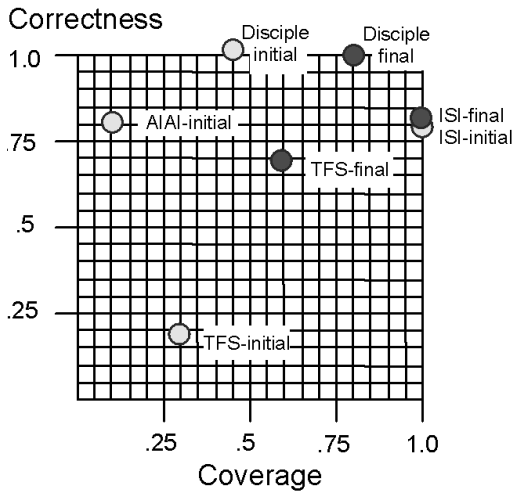

Figure 16. Evaluation results

We entered the evaluation with a workaround agent with a knowledge base which covered only about 40% of the workaround domain (11841 binary predicates). The coverage of our agent was declared prior to each release of the testing problems and all the problems falling within its scope were attempted and scored. During the evaluation period we continued to extend the knowledge base to cover more of the initially specified domain, in addition to the developments required by the modification phase. At the end of the two weeks of evaluation, the knowledge base of our agent grew to cover about 80% of the domain (20324 binary predicates). This corresponds to a rate of knowledge acquisition of approximately 787 binary predicates/day, as indicated in Figure 17. This result supports the claim that the Disciple approach enables rapid acquisition of relevant problem solving knowledge from subject matter experts.

With respect to the quality of the generated solutions, the Disciple agent, within its scope, performed at the level of the human expert. There were several cases during the evaluation period when the Disciple workaround agent generated more correct or more complete solutions than those of the human evaluator. There were also cases when the agent generated new solutions that the human expert did not initially consider. For instance, it generated solutions to work around a cratered road by emplacing a fixed

bridge over the crater in a manner similar to emplacing a fixed bridge over a river gap. Or, in the case of several craters, it generated solutions where some of the craters were filled while fixed bridges were emplaced over others. These solutions were adopted by the evaluator and used as standard for improving all the systems. For this reason, although the Disciple agent also made some mistakes, the overall correctness of its solutions was practically as high as that of the evaluator's solutions. This result supports a second claim that the acquired problem solving knowledge is of a good enough quality to assure a high degree of correctness of the solutions generated by the agent.

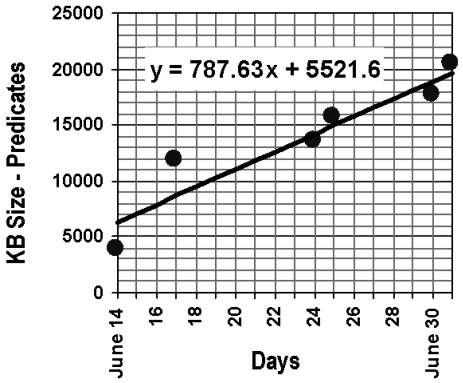

Figure 17. Knowledge acquisition rate

Finally, our workaround generator also had a very good performance, being able to generate a solution in about 0.3 seconds, on a medium power PC. This supports a third claim that the acquired problem solving knowledge assures a high performance of the problem solver.

Based on the evaluation results, the agent developed with Disciple-LAS was selected by DARPA and Alphatech to be further extended and was integrated by Alphatech into a larger system that supports air campaign planning by the JFACC and his/her staff. The integrated system was one of the systems selected to be demonstrated at EFX'98, the Air Force's annual show case of promising technologies.

As compared with Disciple-LAS, the other tools used in the HPKB project to solve the workaround problem reflect a different approach and philosophy to rapid development of knowledge-based systems. ISI’s development environment consists of two domain-independent tools, the LOOM ontology server (MacGregor 1991), and the EXPECT system for knowledge base refinement (Gil 1994), both being tools designed to assist the knowledge engineer, rather than the domain expert. Also, the focus is on

assisting the refinement of the knowledge base rather than its initial creation. The approach taken by both Teknowledge (TFS) and the University of Edinburgh (AIAI) is based on Cyc (Lenat 1995) and emphasizes rapid development of knowledge-based systems through the reuse of the previously developed Cyc ontology. A main difference from our approach is that Cyc is based on a very carefully engineered general ontology, that is to be reused for different applications, while in our case we take the position that the imported ontology should be customized for the current domain. Also, Cyc's concepts and axioms are manually defined, while in Disciple the rules are learned and refined by the system through an interaction with the user.

7. CONCLUSION

In this chapter we have discussed several issues regarding the design and development of intelligent agents, and solutions for these issues in the Disciple approach. The Disciple approach facilitates the development of intelligent agents by domain experts by reducing the complexity of the operations involved in building agents to simpler ones. For instance, rather than developing an entire ontology from scratch, it facilitates importing relevant parts from external knowledge servers. Rather than manually defining and testing task reduction rules, it can learn such rules from specific examples of task reductions and their explanations. Even the definition of an explanation is facilitated by the use of hints and analogical reasoning.

There are, however, several weaknesses in this approach that we plan to address in the future. For instance, the Disciple shell does not yet support the initial modeling of the domain, which is critical to the successful development of the agent. We therefore plan to develop a modeling tool that will use abstract descriptions of tasks and objects in a scenario similar to that used in teaching the agent. Also, importing concepts and features from previously developed ontologies, although very appealing is actually quite hard to accomplish. We are therefore planning to develop methods where the modeling process and the agent provide more guidance in identifying the knowledge pieces to import. Finally, we need to develop a more powerful and natural approach to hint specification by the expert. The current types of allowable hints do not constrain the search for explanations enough, and some of them are not very intuitive for the expert.

Nevertheless, we believe that through such an approach it will some day be possible to develop learning agent shells that will be customized, taught and trained by normal users as easily as they now use personal computers for text processing or email. Therefore, the work on Disciple is part of a long

term vision where personal computer users will no longer be simply consumers of ready-made software, as they are today, but also developers of their own software assistants.

ACKNOWLEDGEMENTS

Andrei Zaharescu has contributed to the development of the interface of Disciple.This research was supported by the AFOSR grant F49620-97-1-0188, as part of the DARPA's High Performance Knowledge Bases Program.

REFERENCES

Alphatech, Inc. 1998. HPKB Year 1 End-to-End Battlespace Challenge Problem Specification, Burlington, MA.

Chandrasekaran, B., and Johnson, T. R. 1993. Generic Tasks and Task Structures: History, Critique and New Directions, In David, J.M., Krivine, J.P., and Simmons, R. eds. Second Generation Expert Systems, pp.239-280. Springer-Verlag.

Chaudhri, V. K., Farquhar, A., Fikes, R., Park, P. D., and Rice, J. P. 1998. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In Proc. AAAI-98, pp. 600 – 607, Menlo Park, CA: AAAI Press.

Clancey, W. J. 1984. NEOMYCIN: Reconfiguring a rule-based system with application to teaching. In Clancey W. J. and Shortliffe, E. H., eds. Readings in Medical Artificial Intelligence, pp.361-381. Reading, MA: Addison-Wesley.

Cohen P., Schrag R., Jones E., Pease A., Lin A., Starr B., Gunning D., and Burke M. 1998. The DARPA High-Performance Knowledge Bases Project, AI Magazine, 19(4),25-49.

Farquhar, A., Fikes, R., and Rice, J. 1996. The Ontolingua Server: a Tool for Collaborative Ontology Construction. In Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada.

Fellbaum, C. ed. 1998. WordNet: An Electronic Lexical Database , MIT Press.

Giarratano, J., and Riley, G. 1994. Expert Systems: Principles and Programming, Boston, PWS Publ. Comp.

Gil, Y. 1994. Knowledge Refinement in a Reflective Architecture. In Proc. AAAI-94, Seattle, WA.

Lenat, D. B. 1995. CYC: A Large-scale investment in knowledge infrastructure Comm of the ACM 38(11):33-38.

MacGregor R. 1991. The Evolving Technology of Classification-Based Knowledge Representation Systems. In Sowa, J. ed. Principles of Semantic Networks: Explorations in the Representations of Knowledge, pp. 385-400. San Francisco, CA: Morgan Kaufmann.

Michalski, R. S. and Tecuci, G., eds. 1994. Machine Learning: A Multistrategy Approach Volume 4. San Mateo, CA.: Morgan Kaufmann.

Musen, M.A. and Tu S.W. 1993. Problem-solving models for generation of task-specific knowledge acquisition tools. In Cuena J. ed. Knowledge-Oriented Software Design, Elsevier, Amsterdam.

Stoffel, K., Taylor, M., and Hendler, J. 1997. Efficient Management of Very Large Ontologies. In Proc. AAAI-97, Menlo Park, Calif.: AAAI Press.

Tecuci, G. 1998. Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies. London, England: Academic Press.

Tecuci, G., Boicu, M., Wright, K., Lee, S.W., Marcu, D. and Bowman, M. 1999. An Integrated Shell and Methodology for Rapid Development of Knowledge-Based Agents. To appear in *Proc. AAAI-99*, July 18-22, Orlando, Florida, Menlo Park, CA:AAAI Press.

Tecuci, G. and Kodratoff, Y. eds. 1995. Machine Learning and Knowledge Acquisition: Integrated Approaches.: Academic Press.

UMLS 1998. Unified Medical Language System, UMLS Knowledge Sources 9th Edition, National Library of Medicine. (http://www.nlm.nih.gov/research/umls/)