In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98) Workshop on "Software Tools for Developing Agents".
AAAI Workshop Technical Report WS-98-10, pp. 37-46, July 26-30, Madison, Wisconsin, AAAI Press, Menlo Park, CA. 1998.

A Learning Agent Shell and Methodology for Developing Intelligent Agents

G. Tecuci, K. Wright, S.W. Lee, M. Boicu, and M. Bowman

Department of Computer Science, MSN 4A5, George Mason University, Fairfax, VA 22030
{tecuci, kwright, swlee, mboicu, mbowman3}@gmu.edu

Abstract

Disciple is a learning agent shell and methodology for efficient development of personal agents. The process of developing an agent with Disciple relies on importing ontologies from existing repositories of knowledge and on teaching the agent to perform various tasks in a way that resembles how an expert would teach an apprentice, by giving the agent examples and explanations, and by supervising and correcting its behavior. The paper presents briefly the architecture of Disciple, the process of developing a Disciple agent, and various agents developed with Disciple. Then it discusses several general issues in the design and development of intelligent agents, and how they are dealt with in Disciple.

Introduction

For several years we have been developing the Disciple apprenticeship, multistrategy learning approach for building intelligent agents (Tecuci, 1998). The defining feature of the Disciple approach to building agents is that a person teaches the agent how to perform domain-specific tasks. This teaching of the agent is done in much the same way as teaching a student or apprentice, by giving the agent examples and explanations, as well as supervising and correcting its behavior. We claim that the Disciple approach significantly reduces the involvement of the knowledge engineer in the process of building an intelligent agent, most of the work being done directly by the domain expert. In this respect, 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.

Architecture of the Disciple Shell

The current version of the Disciple approach is implemented in the Disciple Learning Agent Shell (Tecuci, 1998). We define a *learning agent shell* as consisting of a learning engine and an inference engine that support a representation formalism in which a knowledge base can be encoded, as well as a methodology for building the knowledge base. The architecture of the Disciple shell is presented in Figure 1.

The Disciple shell consists of the five main components in the light gray area which are domain independent:

- a knowledge acquisition and learning component for developing and improving the knowledge base, with a general graphical user interface to enable the expert to interact with the shell for the purpose of developing the knowledge base;
- a knowledge import/export component for accessing remote ontologies located on servers supporting the OKBC protocol suite (Chaudhri et al, 1997);
- a basic problem solving component which serves both to provide the various facilities used by the knowledge acquisition and learning component and to support basic agent operations;
- a knowledge base manager which controls access and updates to the knowledge base; and
- an initial domain-independent knowledge base to be developed for the specific application domain.

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

- a graphical user interface which supports specialized knowledge elicitation and agent operation, determined in part by the nature of the tasks to be performed and in part by the objects in the domain;
- a problem solving component which provides the specific functionality of the Agent.

The domain-specific problem solving component is built on top of the domain independent problem solving operations of the Disciple shell, forming together the Inference Engine of a specific agent. The domain specific interface is also built for the specific agent to allow the domain experts to communicate with the agent as close as possible to the way they communicate in their environment. Some specific agents need additional user interfaces to support unique domain requirements for knowledge representation.

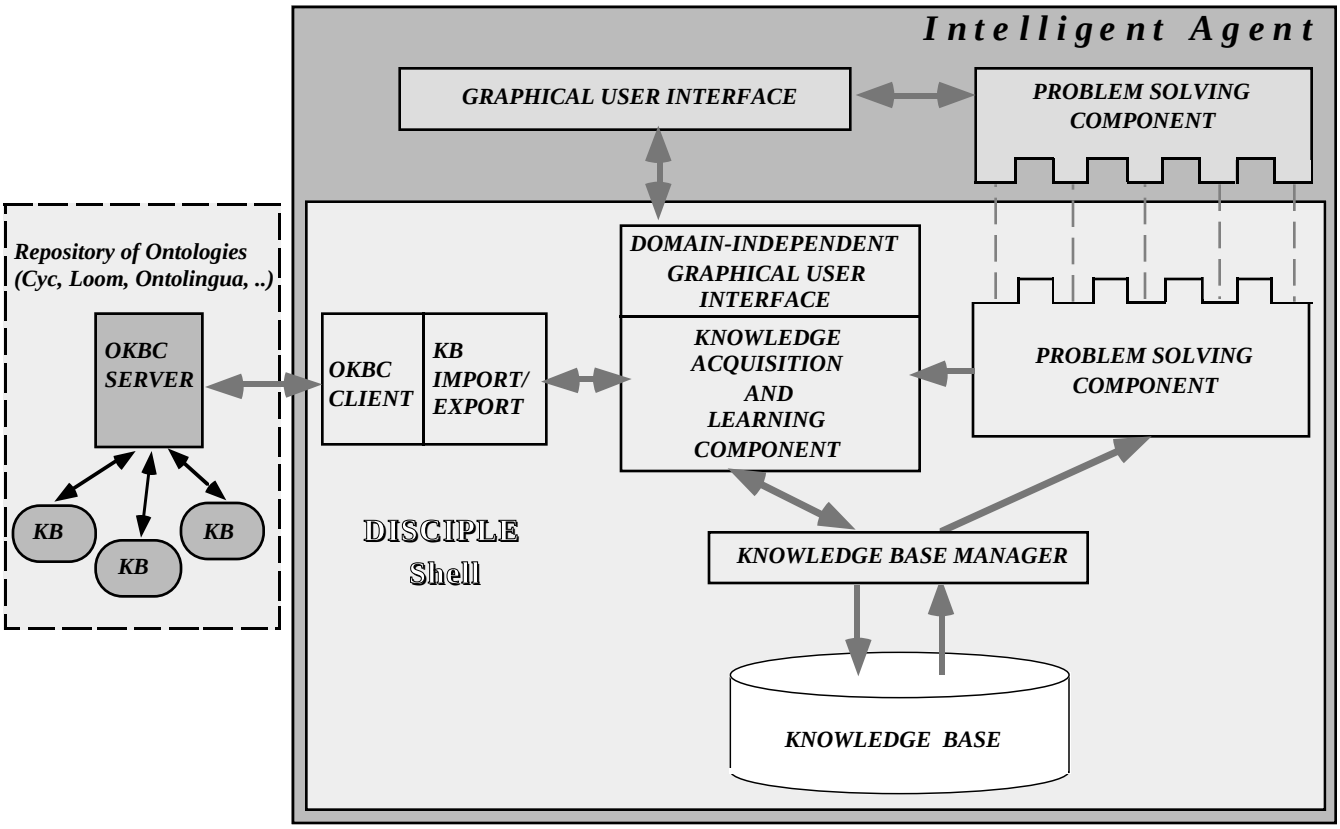

Figure 1 General architecture of the Disciple Learning Agent Shell

The methodology for building agents

The Disciple Learning Shell allows rapid development of a customized agent for a particular application domain. There are two main types of agent which can be developed:

- an agent which is used by and serves as an assistant to the expert, and
- an agent which is used by and provides a service to a non-expert user.

In the first case, the expert is both the developer and the user of the system. The expert initially teaches the agent basic knowledge about the application domain then the agent is used as an assistant by the expert. The agent continues to improve its knowledge during its interactions with the expert. In the second case, the agent is initially developed by the expert then is delivered to the non-expert users. Any necessary changes to its knowledge or operation mean further involvement by the expert.

An overview of the Disciple methodology is given in Figure 2. Depending upon the complexity of the application domain, the expert may require the assistance of a separate developer (a software and/or knowledge engineer) to build the domain-dependent modules of the agent shown in the dark gray area of Figure 1. The expert and the developer should work closely together to determine the customization requirements and build the agent. The dark arrows in Figure 2 indicate the extent of the agent building process if no customization is required but only the development of the knowledge base. The gray arrows highlight the customization activities required during the agent building process. Broken arrows indicate where the agent development process may require some additional development effort after the agent has been put to use, either due to changes in the application domain or to changes in the operational requirements for the agent.

There are three stages and three different participants in the agent's lifetime:

1 The agent developer (software and knowledge engineer), cooperating with the domain expert, customizes the Disciple shell by developing a domain specific interface on top of the domain independent graphical user interface of the Disciple shell. This domain specific interface gives the domain experts a natural means of expressing their knowledge. The result of this effort is an agent which has learning capabilities, a generic problem solving component, and an empty knowledge base, but with an interface customized for knowledge elicitation. This agent can interact with the expert during the knowledge base development process.

The domain expert and the agent developer also decide on the nature and extent of a domain-specific problem solver, based upon the type and purpose of the agent to be developed. The agent developer continues the

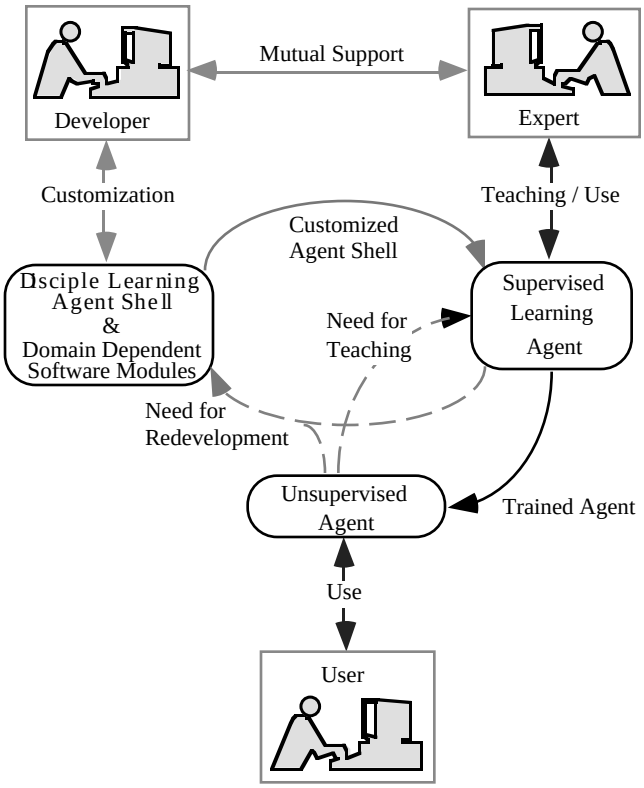

Figure 2 Disciple's agent-building methodology

customization of the agent with the development of the domain-specific problem solver. The final agent is fully customized and can interact with the expert not only for knowledge elicitation but also for problem solving and learning.

2 The domain expert interacts with the customized agent to develop its initial knowledge base and to teach it to perform domain specific tasks. The expert both teaches and uses an agent which is to be his or her assistant. If the agent is to be used by another user, the expert teaches, verifies and validates the agent and then releases it to the user.

3 The agent is released to the user and performs the tasks it was taught. In some cases the agent may require retraining by the expert or further customization by the developer to update or refine its interfaces or problem solving methods. The process of retraining and redevelopment does not differ from the agent's initial training and development.

In summary, the building of a Disciple agent consists of the following activities:

- the customization of the Disciple shell to support specialized knowledge elicitation;
- development of the agent's ontology and its representation in a semantic network;
- customization of the Disciple shell with a domain-dependent problem solver;
- training of the agent for its domain-specific tasks; and
- verification and validation of the agent.

Application domains

We have experimentally applied the Disciple methodology and shell to the development of five different agents.

Two of the agents generate history tests to assist in the assessment of students' understanding and use of higher-order thinking skills (Tecuci and Keeling, 1998). These two assessment agents are representative of the class of agents built by an expert (in education and history, in this case) to assist other users (history teachers and students). One of the assessment agents is integrated with the MMTS (Multimedia and Thinking Skills) educational system, creating a system with expanded capabilities, called Intelligent MMTS (IMMTS). Inside IMMTS, the agent has the role of generating an exam consisting of a set of test questions of different levels of difficulty. The student has to answer one test question at a time and, after each question, he or she receives the correct answer and an explanation of the answer. This type of application of a Disciple-generated agent illustrates a possible role for these agents, that of enhancing the capabilities, generality, and usefulness of non-KB software. The IMMTS system has been field-tested in American history classes in several middle schools on American installations in Germany and Italy. The other test generation agent is a stand-alone agent that can be used independently of the MMTS software. The student interacts directly with this agent to assess him/herself. He or she chooses the type of test question to solve, and will receive, on request, feedback in the form of hints to answer the question, the correct answer, and some or all the explanations of the answer. That is, this agent also tutors the student. We have performed four types of experiments with the stand-alone test generation agent, and the results were very encouraging. The first experiment tested the correctness of the agent's knowledge base, as judged by the domain expert who developed the agent. This was intended to clarify how well the developed agent represents the expertise of the expert who taught the agent. The second experiment tested the correctness of the knowledge base, as judged by a domain expert who was not involved in its development. This was intended to test the generality of the agent. The third and the fourth experiments tested the quality of the test generation agent, as judged by students and by teachers.

Another agent, the Statistical Analysis Assessment and Support Agent, is developed to be integrated in a university-level introductory science course and to be accessed on the Internet through a web browser. The course "The Natural World" introduces students to the world of science using collaborative assignments and problem-centered group projects that look at scientific issues which underlie public policy making and stimulate the development of students' analytic skills. The agent supports two aspects of students' learning in this course: students' knowledge and understanding of statistics, and

students' analyses of issues related to statistics. It does this in several ways. As in the case of the two history assessment agents, it can be used as a traditional test generator. It also integrates the documents accessed by students on the web and interacts with the students during the learning process. Finally, it can be used as an assistant by the students as they work through their assignments.

An early agent was developed by an expert to act as an assistant in an engineering design domain by supporting computer configuration tasks (Dybala et al., 1996). This type of agent has to be continuously supervised and customized by the user according to the changing practices in the user's domain, as well as the needs and the preferences of the user. The agent initially behaves as a novice unable to compose the majority of designs. As the expert and the assistant interact, the assistant learns to perform most of the routine (but usually more labor intensive) designs within the domain. Because of its plausible reasoning capabilities, the assistant is also able to propose innovative designs that are corrected and finalized by the designer. Creative designs are specified by the designer and presented to the assisting agent. As a result of learning, designs that were innovative for the assistant became routine, and designs that were creative became first innovative and later routine ones.

Another early agent is an agent trained to behave as a military commander in a virtual environment (Tecuci and Hieb, 1996). This is a type of agent that is trained by a user to perform tasks on user's behalf. The virtual military environment is the ModSAF (Modular Semi-Automated Forces) distributed interactive simulation that enables human participants at various locations to enter a synthetic world containing the essential elements of a military operation (Ceranowicz, 1994). ModSAF is a very complex real-time application which simulates military operations. In the ModSAF environment, human participants may cooperate with, command or compete against virtual agents. The agent was trained to perform defensive missions using the graphical interface of ModSAF.

One general conclusion that can be drawn from these experimental applications is that the Disciple approach could be easily used to develop agents for a wide range of problems and domains.

The rest of this paper discusses the Disciple proposed solutions to several critical issues that have been found to be limiting factors in building intelligent agents for complex real world domains.

General issues in developing Disciple agents

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:

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

Each of these issues has been an important concern in developing the Disciple agent building approach. We discuss each of them in the following.

Reuse of previously developed knowledge

Sharing and reusing the components of different Knowledge Representation Systems are hard research problems because of the incompatibilities in their implicit knowledge models (the precise definition of declarative knowledge structures) assumed by their various underlying knowledge components. Recently, however, the Open Knowledge Base Connectivity (OKBC) protocol (formerly called "Generic Frame Protocol") has been developed. OKBC is a standard for accessing knowledge bases stored in different frame representation systems (Chaudhri et al, 1997). It provides a set of operations for a generic interface to such systems. There is also an ongoing effort of developing OKBC servers, such as the Cyc (Lenat, 1995), Loom (MacGregor, 1995), and Ontolingua (Farquhar et al., 1996) servers. These servers are becoming repositories of reusable ontologies and domain theories, and can be accessed using the OKBC protocol.

The knowledge base of a Disciple agent consists of an ontology (Gruber, 1993) that defines and organizes the concepts from the application domain, and a set of problem solving rules expressed in terms of these concepts. The process of building this knowledge base starts with creating a domain ontology, by accessing an OKBC server and importing concepts from the available shared ontologies. To make this possible, we are currently developing a OKBC wrapper, to make Disciple an OKBC client. We are also developing the "Knowledge Import/Export Module" which allows an expert to guide the import of knowledge from an OKBC server, as well as to export knowledge from the Disciple's knowledge base. For example, the domain expert can extract some interesting concepts from server and represent them in Disciple's KB. During this process, the expert can freely modify the definitions of the imported terms. The expert can also extract an entire sub-hierarchy of a certain concept and the knowledge import module will automatically introduce this new knowledge into Disciple's knowledge base. This process involves various kinds of verifications to maintain the consistency of Disciple's knowledge.

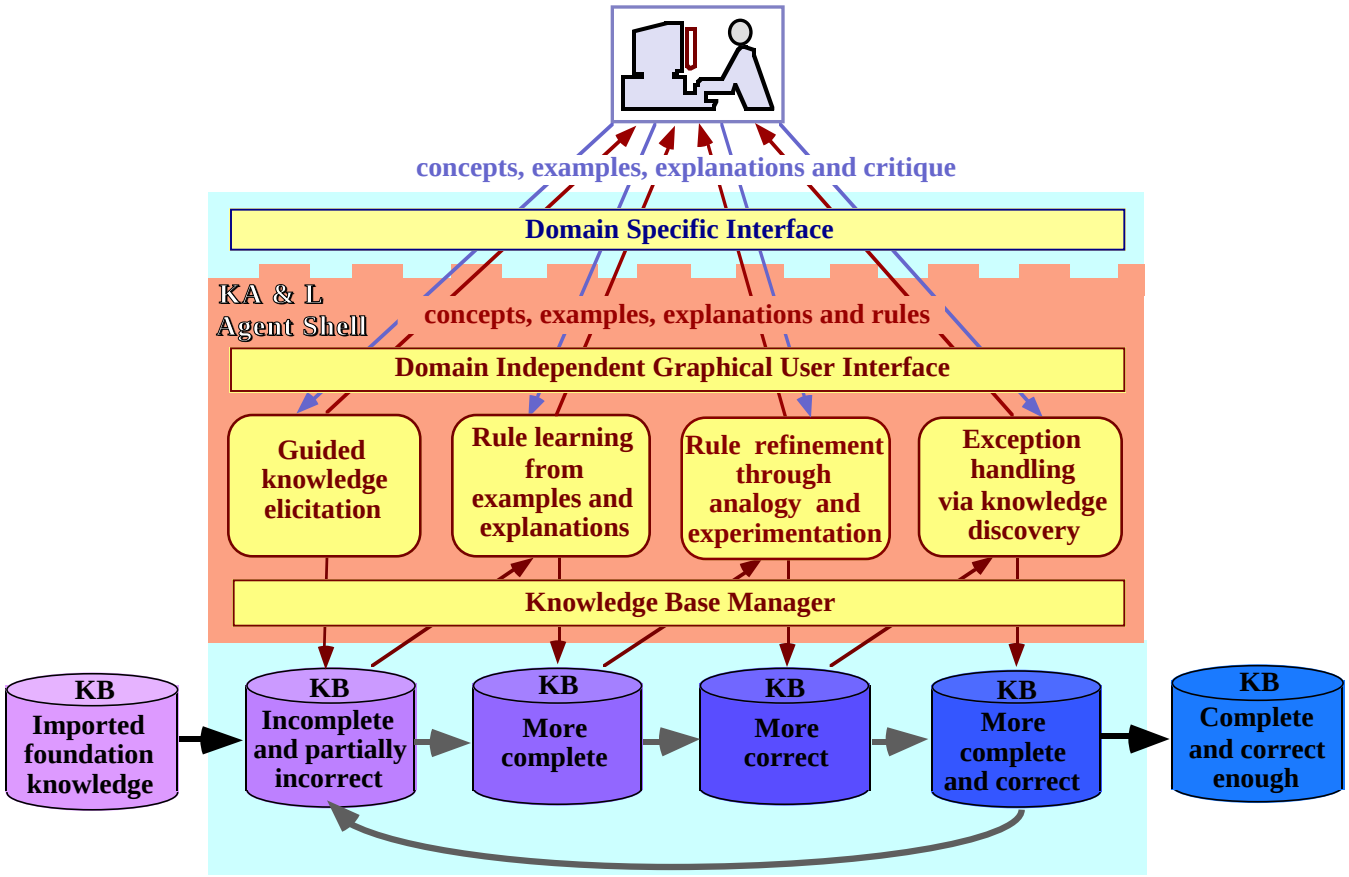

Figure 3 Processes supporting the development of the knowledge base

The knowledge acquisition bottleneck

The knowledge acquisition bottleneck expresses the difficulty of encoding knowledge in the knowledge base of an intelligent agent. The Disciple approach is developed to primarily address this issue.

Figure 3 presents the knowledge base development processes which are supported by Disciple's knowledge acquisition and learning component (see also Figure 1).

During *Knowledge Elicitation*, the expert defines knowledge that he/she could easily express. As has been indicated in the previous section, some of the initial knowledge could also be imported from an existing knowledge base.

During *Rule Learning*, the expert teaches the agent how to solve domain specific problems. He/she shows the agent how to solve typical problems and helps it to understand their solutions. The agent uses learning from explanations and by analogy to learn general plausible version space rule that will allow it to solve similar problems.

During *Rule Refinement*, the agent employs learning by experimentation and analogy, inductive learning from examples and learning from explanations, to refine the rules in the knowledge base. These could be either rules learned during the Rule Learning process, rules directly defined by the expert during Knowledge Elicitation, or rules that have other origins (for instance, rules transferred from another knowledge base). Rule refinement will also cause a refinement of the concepts from the agent's ontology (semantic network).

A refined rule may have exceptions. A negative exception is a negative example that is covered by the rule and a positive exception is a positive example that is not covered by the rule. One common cause of the exceptions is the incompleteness of the knowledge base; that is, it does not contain the terms to distinguish between the rule's examples and exceptions. During *Exception Handling*, the agent hypothesizes additional knowledge and/or guides the expert to define this knowledge in agent's ontology. This will extend the representation space for learning such that, in the new space, the rules could be modified to remove the exceptions.

The expert interacts with the agent's knowledge acquisition and learning facilities via both a domain-dependent interface and a domain-independent interface. These interfaces provide overall control over the process of developing the knowledge base, and consists of browsing and editing facilities, which support the knowledge elicitation requirements for building an ontology, and rule learning and refinement facilities, which integrate multistrategy machine learning techniques with the expert's experience for the development of rules.

We will briefly illustrate part of this process with an example of teaching a workaround agent. The agent has to act as an assistant of anlysist who has to determine the best way of working around various damages to an infrastructure, such as a damaged bridge or tunnel.

Rule Learning. The rule learning method is schematically represented in Figure 4. As Explanation-based Learning (DeJong and Mooney, 1986; Mitchell, Keller, Kedar-Cabelli, 1986), it 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. Figure 5 shows the rule learning interface after a rule has been learned.

The expert starts teaching the agent how one could work around a certain class of infrastructure damage (such as a damaged bridge) by providing an initial example of a workaround task and its solution. A narrative description of the example is dynamically constructed through an interaction between the expert and the agent. An image corresponding to the bridge described in the task is also dynamically displayed for the expert's review before the solution is developed (see the bottom of Figure 5). The internal representation of the example is generated by Disciple and shown in the top left pane of Figure 5.

In the explanation phase (see Figure 4), the expert helps the agent to understand why the example is correct. He or she guides the agent to propose explanations and then selects the correct ones. For instance, the expert may point to the most relevant objects from the input example and may specify the types of explanations to be generated by the agent (e.g. a correlation between two objects or a property of an object). The agent uses such guidance and specific heuristics to propose plausible explanations to the expert who has to select the correct ones. Initially, the expert must select from many proposed explanations. However, as Disciple learns more and more rules within

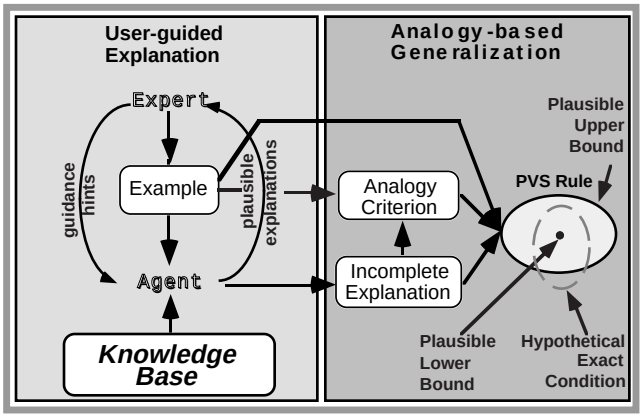

Figure 4 The rule learning method of Disciple.

the domain, it can immediately suggest fewer, possibly relevant explanations via analogical reasoning with these rules. For instance, if the agent "understands" why a company can build a floating bridge over a river segment then, by analogy, it can hypothesize many of the conditions required to use a floating bridge as a ferry, or to simply ford the river. This is very important for rapidly developing large knowledge bases because typically such knowledge bases are composed of clusters of similar rules. Once a rule from a cluster has been learned, the agent can much easier learn the other rules. The middle pain in Figure 5 shows the explanations of the initial example.

Once the explanations are identified, Disciple enters the generalization phase (see Figure 4), and performs an analogy-based generalization of the example and its explanation into a plausible version space (PVS) rule. A PVS rule is an IF-THEN rule except that, instead of a single applicability condition, it has two conditions that represent a plausible version space for the hypothetical exact condition. The plausible lower bound condition is very specific, covering only the given example. The plausible upper bound condition is an analogical generalization of the plausible lower bound condition. Part of the plausible upper bound condition of this rule (in a simplified form) is shown in the upper right of Figure 5.

The explanation is generalized to an analogy criterion by generalizing its components. Object instances are generalized to more general concepts from the agent's ontology. Numbers are generalized to intervals or union of intervals, to functional expressions (such as "(* ?u1 ?n1)") or to symbolic concepts (such as "odd-number"). Strings are generalized to sets of strings. For instance, the third explanation piece from the middle pane of Figure 5

M4T6-TRACK-COMP-D TOTAL-LENGTH 480,
MO-RIVER-UP-647-314 WIDTH-OF-RIVER 300 < 480

which means "M4T6 bridging capability of Tracked Company D has a total length of 480m and the Missouri river segment at map coordinates 647-314 has a width of 300m < 480m", is automatically generalized to

?M17	IS	RIVER-SEGMENT
	WIDTH-OF-RIVER	?M20
?M18	IS	BREACHING-CAPABILITY
	TOTAL-LENGTH	?M19
?M19	IS	[0 , 2000]
	:=	(* ?M27 ?M28)
?M20	IS	[0 , 2000)
	<	?M19

To determine how to generalize an object, Disciple analyzes all the features from the example and the explanation that are connected to that object. Each such feature is defined in Disciple's ontology by a domain (that specifies the set of all the objects from the application domain that may have that feature) and a range (that specifies all the possible values of that feature).

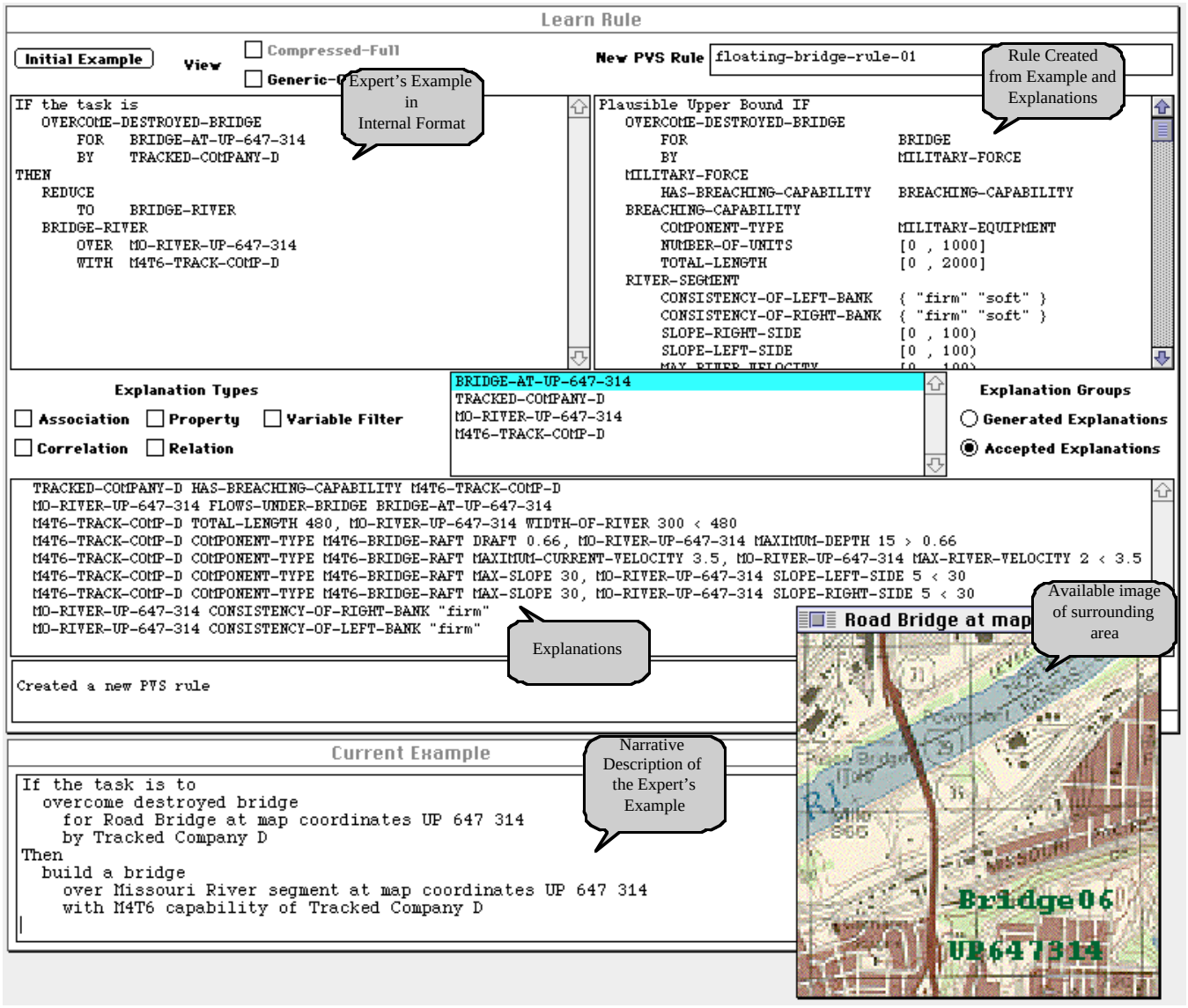

Figure 5 Learning a rule from an initial example and proposed explanations.

The domains and the ranges of these features restrict the generalizations of the objects. The generalization of "480m" to "(* ?M27 ?M28)" is based on the fact that the total length of a floating bridge that a company can build is computed by multiplying the number of floating units and the length of such a unit.

The analogy criterion and the example are used to generate the plausible upper bound condition of the rule, while the explanation and the example are used to generate the plausible lower bound condition of the rule. The learned rule is shown in Figure 6.

Rule refinement. The representation of the PVS rule in the right hand side of Figure 4 shows the most likely relation between the plausible lower bound, the plausible upper bound and the hypothetical exact condition of the rule. Notice that there are instances of the plausible upper bound that are not instances of the hypothetical exact condition of the rule. This means that the learned rule in Figure 6 covers also some negative examples. Also, there are instances of the hypothetical exact condition that are not instances of the plausible upper bound. This means that the plausible upper bound does not cover all the positive examples of the rule. Both of these situations are a consequence of the fact that the explanation of the initial example might be incomplete, and are consistent with what one would expect from an agent performing analogical reasoning. To improve this rule, the expert will invoke the rule refinement process represented schematically in Figure 7. The expert will ask the agent to use the learned rule to generate examples similar with the initial example (the one from the bottom of Figure 5. Each example generated by the agent is covered by the plausible upper bound and is

```
Plausible Upper Bound IF                                    ; IF (Plausible Upper Bound)
   ?O11    IS                      OVERCOME-DESTROYED-BRIDGE ; the task is to overcome a destroyed
           FOR                     ?B16                      ; bridge for ?B16
           BY                      ?T16                      ; by ?T16 and
   ?B16    IS                      BRIDGE                    ; ?B16 is a bridge and
   ?T16    IS                      MILITARY-FORCE            ; ?T16 is a military force that
           HAS-BREACHING-CAPABILITY ?M18                     ; has the breaching capability ?M18 and
   ?M18    IS                      BREACHING-CAPABILITY      ; ?M18 is a breaching capability
           COMPONENT-TYPE          ?M23                      ; with the component type ?M23 and
           NUMBER-OF-UNITS         ?M27                      ; number of units ?M27 and
           TOTAL-LENGTH            ?M19                      ; the total length ?M19 and
   ?M23    IS                      MILITARY-EQUIPMENT        ; ?M23 is a military equipment
           MAX-SLOPE               ?M26                      ; characterized by a maximum slope of ?M26
           MAXIMUM-CURRENT-VELOCITY ?M24                     ; a maximum current velocity of ?M24
           LENGTH-OF-UNIT          ?M28                      ; a length of ?M28
           DRAFT                   ?M21                      ; and a draft of ?M21
   ?M27    IS                      [0 , 1000]                ; number of units ?M27 is between 0 and 1000,
   ?M19    IS                      [0 , 2000]                ; and ?M19 is in the interval [0 , 2000]
           :=                      (* ?M27 ?M28)             ; computed by the product of ?M27 and ?M28
   ?M17    IS                      RIVER-SEGMENT             ; ?M17 is a river segment which
           FLOWS-UNDER-BRIDGE      ?B16                      ; flows under the bridge ?B16 & is characterized
           CONSISTENCY-OF-LEFT-BANK ?M53                     ; by consistency of the left bank ?M53
           CONSISTENCY-OF-RIGHT-BANK ?M47                    ; consistency of the right bank ?M47
           SLOPE-RIGHT-SIDE        ?M43                      ; slope of right side ?M43
           SLOPE-LEFT-SIDE         ?M42                      ; slope of left side ?M42
           MAX-RIVER-VELOCITY      ?M25                      ; maximum river velocity ?M25
           MAXIMUM-DEPTH           ?M22                      ; maximum depth ?M22
           WIDTH-OF-RIVER          ?M20                      ; width ?M20
   ?M20    IS                      [0 , 2000)                ; river width ?M20 is between 0 and 2000m
           <                       ?M19                      ; and is less than ?M19, and
   ?M21    IS                      (0 , 100]                 ; draft ?M21 is between 0m and 100m, and
   ?M22    IS                      (0 , 100]                 ; max depth ?M22 is between 0m and 100m
           >                       ?M21                      ; and is greater than ?M21, and
   ?M24    IS                      (0 , 100]                 ; max c. rate ?M24 is between 0 and 100m/s
   ?M25    IS                      [0 , 100)                 ; max riv.vel ?M25 is between 0 and 100m/s
           <                       ?M24                      ; and is less than ?M24, and
   ?M26    IS                      (0 , 100]                 ; max slope ?M26 is between 0 and 100%,
   ?M28    IS                      (0 , 100]                 ; length of unit ?M28 is between 0 and 100m,
   ?M42    IS                      [0 , 100)                 ; left slope ?M42 is between 0% and 100%
           <                       ?M26                      ; and is less than ?M26, and
   ?M43    IS                      [0 , 100)                 ; right slope ?M43 is between 0% and 100%
           <                       ?M26                      ; and is less than ?M26
   ?M47    IS                      { "firm" "soft" }         ; consistency of right bank ?M47
   ?M53    IS                      { "firm" "soft" }         ; consistency of left bank ?M53
Plausible Lower Bound IF                                    ; IF (Plausible Lower Bound)
   ?O11    IS                      OVERCOME-DESTROYED-BRIDGE ; the task is to overcome a destroyed
           FOR                     ?B16                      ; bridge for ?B16
           BY                      ?T16                      ; by ?T16 and
   ?B16    IS                      BRIDGE-AT-UP-647-314      ; ?B16 is the bridge at map grid UP 647 314 and
   ?T16    IS                      TRACK-COMP-D              ; ?T16 is tracked Company D
           HAS-BREACHING-CAPABILITY ?M18                     ; has the breaching capability ?M18 and
   ?M18    IS                      M4T6-TRACK-COMP-D         ; ?M18 is the M4T6 breaching capability of CompanyD
           COMPONENT-TYPE          ?M23                      ; with the component type ?M23 and
           NUMBER-OF-UNITS         ?M27                      ; number of units ?M27 and
           TOTAL-LENGTH            ?M19                      ; the total length ?M19 and
   ?M23    IS                      M4T6-BRIDGE-RAFT          ; ?M23 is a M4T6 bridging/rafting component
           MAX-SLOPE               ?M26                      ; characterized by a maximum bank slope of ?M26
           MAXIMUM-CURRENT-VELOCITY ?M24                     ; a maximum current velocity of ?M24
           LENGTH-OF-UNIT          ?M28                      ; a length of ?M28
           DRAFT                   ?M21                      ; and a draft of ?M21
   ?M27    IS                      { 24 }                    ; number of units ?M27 is 24,
   ?M19    IS                      { 480 }                   ; and total length ?M19 is 480 meters
           :=                      (* ?M27 ?M28)             ; computed by the product of ?M27 and ?M28
   ?M17    IS                      MO-RIVER-UP-647-314       ; ?M17 is the Missouri river segment which
           FLOWS-UNDER-BRIDGE      ?B16                      ; flows under the bridge ?B16 and is characterized
           CONSISTENCY-OF-LEFT-BANK ?M53                     ; by consistency of the left bank ?M53
           CONSISTENCY-OF-RIGHT-BANK ?M47                    ; consistency of the right bank ?M47
           SLOPE-RIGHT-SIDE        ?M43                      ; slope of right side ?M43
           SLOPE-LEFT-SIDE         ?M42                      ; slope of left side ?M42
           MAX-RIVER-VELOCITY      ?M25                      ; maximum river velocity ?M25
           MAXIMUM-DEPTH           ?M22                      ; maximum depth ?M22
           WIDTH-OF-RIVER          ?M20                      ; width ?M20
   ?M20    IS                      { 300 }                   ; river width ?M20 is 300 meters
           <                       ?M19                      ; and is less than ?M19, and
   ?M21    IS                      { 0.66 }                  ; draft ?M21 is 0.66 meters,
   ?M22    IS                      { 15 }                    ; max depth ?M22 is between 15 meters
           >                       ?M21                      ; and is greater than ?M21, and
   ?M24    IS                      { 3.5 }                   ; max c. rate ?M24 is 3.5 meters/second
   ?M25    IS                      { 2 }                     ; max riv. vel. ?M25 is 2 meters/second
           <                       ?M24                      ; and is less than ?M24, and
   ?M26    IS                      { 30 }                    ; max slope ?M26 is 30%, and
   ?M28    IS                      { 20 }                    ; length of unit ?M28 20 meters,
   ?M42    IS                      { 5 }                     ; left slope ?M42 is 5%
           <                       ?M26                      ; and is less than ?M26, and
   ?M43    IS                      { 5 }                     ; right slope ?M43 is 5%
           <                       ?M26                      ; and is less than ?M26, and
   ?M47    IS                      { "firm" }                ; consistency of right bank ?M47 is firm
   ?M53    IS                      { "firm" }                ; consistency of left bank ?M53 is firm
THEN                                                        ; Then
   REDUCE  TO                      ?B12                      ; Reduce this task to ?B12, where
   ?B12    IS                      BRIDGE-RIVER              ; ?B12 is the task to Bridge river
           OVER                    ?M17                      ; over the river segment ?M17
           WITH                    ?M18                      ; with the breaching capability ?M18
```

Figure 6 Learned floating bridge rule

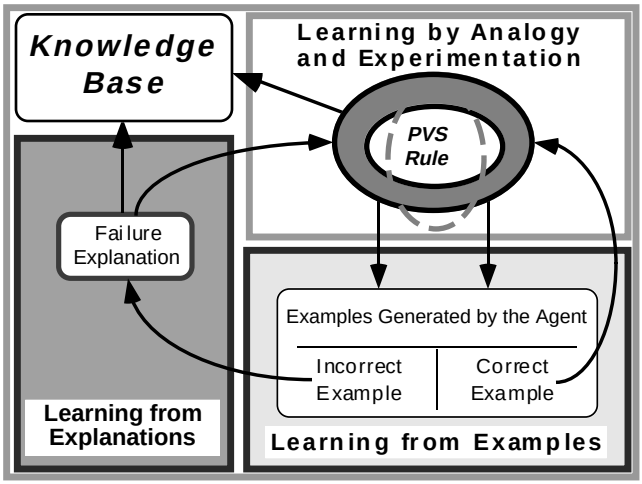

Figure 7 The rule refinement method of Disciple

not covered by the plausible lower bound of the rule. The natural language equivalent of this example (which looks like the one in Figure 5) is shown to the expert who is asked to accept it as correct or to reject it, thus characterizing it as a positive or a negative example of the rule. A positive example is used to generalize the plausible lower bound of the rule's condition through empirical induction. A negative example is used to elicit additional explanations from the expert and to specialize both bounds, or only the plausible upper bound. During rule refinement the two conditions of the rule will converge toward the exact applicability condition.

The collaboration between the expert and Disciple during knowledge elicitation and rule learning is an iterative process which ultimately results in a knowledge base which is complete and correct enough for the successful operation of the agent.

The central idea of the Disciple approach is to facilitate the agent building process by the use of synergism at several levels. First, there is the synergism between different learning methods employed by the agent. By integrating complementary learning methods (such as inductive learning from examples, explanation-based learning, learning by analogy, learning by experimentation) in a dynamic way, the agent is able to learn from the human expert in situations in which no single strategy learning method would be sufficient. Second, there is the synergism between teaching (of the agent by the expert) and learning (from the expert by the agent). For instance, the expert may select representative examples to teach the agent, may provide explanations, and may answer agent's questions. The agent, on the other hand, will learn general rules that are difficult to be defined by the expert, and will consistently integrate them into its knowledge base.

The knowledge adaptation bottleneck

The knowledge adaptation bottleneck expresses the difficulty of changing the knowledge in the knowledge base of the agent in response to changes in the application domain or in the requirements of the agent. While a natural approach to this type of bottleneck would be autonomous learning by the agent from its own experience, this approach alone is not powerful enough for complex application environments. Disciple supports an approach to the knowledge adaptation bottleneck based on a retraining process. During its normal problem solving activity, the Disciple agent will accumulate exceptions to general rules and concepts. Also, various changes in the application domain will require corresponding updates of the knowledge base. When the mismatch between the agent's model of the world and the world itself is above a certain threshold, the agent enters a retraining phase in which it is again directly taught by the expert. This *Retraining Process* involves the same processes of Knowledge Elicitation, Rule Learning, Rule Refinement and Exception Handling**.** That is, in the Disciple approach, knowledge maintenance over the life-cycle of the knowledge base is no different from knowledge acquisition. Indeed, because the whole process of developing the knowledge base is one of creating and adapting knowledge pieces, this creation and adaptation may also occur in response to changes in the environment or goals of the system.

The scalability of the agent building process

Another critical issue addressed by the Disciple approach is the scalability of the agent building process. This is mainly achieved in two ways. The first is the use of an advanced model of interaction between the expert and the agent that allows the expert to guide the agent in building a large knowledge base. The second is the use of efficient multistrategy learning methods based on the plausible version space representation (Tecuci and Hieb, 1996).

Balance between using general tools and developing domain specific modules

In designing an agent building tool, it is also important to find a suitable balance between using general (and therefore reusable) modules and specific (and therefore powerful) modules. Using 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. The Disciple shell provides a set of general and powerful modules for knowledge acquisition and learning. They are domain-independent and are incorporated as such in a developed agent. However, for the interface and the problem solver, that are domain dependent, the Disciple shell contains a generic graphical-user interface and problem solving modules that support only basic problem solving operations (such as, transitivity of certain relations, inheritance of features in a semantic network, network matching, rule matching and example generation). Therefore, for a given application domain, one has to develop additional, domain-specific interfaces and

problem solving modules, in order to create an easy to train and a useful agent. Moreover, if the agent has to execute in, or communicate with, an existing application, such as the MMTS or ModSAF, then one also has to develop the interface with the application. For instance, Figure 4 shows both a domain-independent graphical user interface (the top window), and two domain dependent interfaces (the bottom windows). The problem solver for this workaround agent is also specially developed for it. It is a problem solver based on problem decomposition which is implemented on top of the basic problem solving operations of the Disciple shell.

The portability of the agent building tools and of the developed agents

Currently, the Disciple shell is implemented in Common Lisp and runs on Macintosh. However, only the interface is platform dependent. To enable the use of Disciple on a variety of platforms, to allow future development of multiple user and networked access to Disciple, and to simplify the creation of specialized interfaces, Disciple's interfaces are currently being developed as a JAVA-based graphical user interface having a client-server relationship with the other components of Disciple.

Conclusions

In this paper we have discussed several issues in the design and development of intelligent agents, and their solutions in the Disciple approach. 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.

Acknowledgments. This research was done in the Learning Agents Laboratory. The research of the Learning Agents Laboratory is supported by the AFOSR grant F49620-97-1-0188, as part of the DARPA's High Performance Knowledge Bases Program, by the DARPA contract N66001-95-D-8653, as part of the Computer-Aided Education and Training Initiative, and by the NSF grant No. CDA-9616478, as part of the Collaborative Research on Learning Technologies Program .

References

Bradshaw, J. M. ed. 1997. *Software Agents*. Menlo Park, CA.: AAAI Press.

Buchanan, B. G. and Wilkins, D. C. eds. 1993. *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems,* San Mateo, CA.: Morgan Kaufmann.

Ceranowicz, A. 1994. ModSAF Capabilities. In Proceedings of the 4th Conference on Computer Generated Forces and Behavior Representation. May. Orlando, FL.

Chaudhri, V. K., Farquhar, A., Fikes, R., Karp, P., and Rice, J. 1997. Open Knowledge Base Connectivity 2.0. Technical Report, Artificial Intelligence Center of SRI International and Knowledge Systems Laboratory of Stanford University.

DeJong, G. and Mooney, R. 1986. Explanation-Based Learning: An Alternative View. *Machine Learning* 1:145-176.

Dybala, T., Tecuci, G. and Rezazad, H., 1996. The Shared Expertise Model for Teaching Interactive Design Assistants. *Journal of Engineering Applications of Artificial Intelligence* 9(6):611-626.

Farquhar, A., Fikes, R., and Rice, J. 1996. The Ontolingua Server: a Tool for Collaborative Ontology Construction. In Proceedings of the Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW '96), Banff, Alberta, Canada.

Gruber, T. R. 1993. Toward principles for the design of ontologies used for knowledge sharing. In Guarino, N. and Poli, R. eds. *Formal Ontology in Conceptual Analysis and Knowledge Representation.*: Kluwer Academic.

Lenat, D. B. and Guha, R. V. 1990. *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Readings, Mass.: Addison-Wesley.

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

MacGregor R. et al. 1995. The LOOM Tutorial, http://www.isi.edu/isd/LOOM/documentation/tutorial2.1.ps

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

Mitchell, T. M., Keller, T., and Kedar-Cabelli, S. 1986. Explanation-Based Generalization: A Unifying View. *Machine Learning* 1:47-80.

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

Tecuci, G. and Hieb, M. H. 1996. Teaching Intelligent Agents: The Disciple Approach. *International Journal of Human-Computer Interaction* 8(3):259-285.

Tecuci G. and Keeling H. 1998. Teaching an Agent to Test Students. In *Proceedings of ICML'98*.: Morgan Kaufmann.

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