In *Proceedings of AAAI-2004 Workshop on Intelligent Agent Architectures: Combining the Strengths of Software Engineering and Cognitive Systems,* July 26, San Jose, AAAI Press, Menlo Park, CA, 2004.

Design Principles for Learning Agents

Mihai Boicu, Gheorghe Tecuci, Bogdan Stanescu, Dorin Marcu, Marcel Barbulescu, Cristina Boicu

MSN 4A5, Learning Agents Center, George Mason University, Fairfax, VA 22030, USA
{mboicu, tecuci, bstanesc, dmarcu, ccascava}@gmu.edu, mbarb@cs.gmu.edu, http://lalab.gmu.edu

Abstract

This paper presents several design principles used in the development of the Disciple learning agents. The process of developing such an agent relies on importing ontologies from existing knowledge repositories, 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. Experimental results support the usefulness of the presented principles which may be useful for the development of other agents.

Introduction

For almost 20 years we have performed research on developing the Disciple theory and the associated methodologies and tools for building agents that incorporate the knowledge of a subject matter expert (Tecuci 1988, 1998; Boicu 2002). The long term goal of this research is to create a type of agent that can be taught directly by a subject matter expert (SME), in a way that resembles how the SME would teach a human apprentice when solving problems in cooperation. For instance, the SME may teach the agent by providing it examples on how to solve specific problems, by helping it to understand the solutions, and by supervising and correcting the problem solving behavior of the agent. Our claim is that the Disciple approach will lead to rapid development and easy maintenance of knowledge-based agents, by SMEs and typical computer users, who will require only limited assistance from knowledge engineers.

General research on the development of expert and knowledge-based systems (Buchanan and Wilkins 1993), as well as our own research, has led to the identification of several design principles, which we have successfully used in the Disciple approach. This paper states these principles and illustrates them in Disciple-RKF, the implementation of the most recent version of the Disciple approach. Our claim is that these principles (some of which are already well recognized) are useful for the design of other types of agents.

The architecture of Disciple-RKF is presented in the next section, illustrating three architectural design principles. The first is the separation between a domain-independent agent shell, and the domain-specific modules, which together constitute a knowledge-based agent customized for a specific application. The second principle is the separation between the modules dedicated to different cognitive functions, such as problem solving, learning, and memory. The third principle is the implementation of the modules as a group of collaborative agents.

The section Problem Solving presents the problem solving component of Disciple-RKF. The corresponding design principle is the use of a general problem solving paradigm that is applicable to a wide range of expertise domains, such as planning or critiquing.

The follow-on section discusses the organization of the knowledge base, and illustrates two additional design principles. The first is the separation between the ontology that defines the concepts from an application domain, and the actual problem solving rules (or methods) that are expressed with these concepts. The second principle relates to the representation and use of the partially learned knowledge.

The paper continues with the presentation of the knowledge acquisition and learning component of Disciple, and illustrates three additional design principles: 1) Integration of mixed-initiative problem solving and learning (where the SME and the agent solve problems in cooperation and the agent learns from the problem solving contributions of the SME); 2) Integration of teaching and learning (where the agent helps the SME to teach it, by asking relevant questions, and the SME helps the agent to learn, by providing examples, hints and explanations); and 3) Multistrategy learning (where the agent uses multiple strategies, such as learning from examples, from explanations, and by analogy, to learn general concepts and rules).

The subsequent section presents the end-to-end methodology of developing a Disciple agent, which is based on the above principles. Finally, the last section presents some evaluation results that support these principles.

The Architecture of a Disciple Agent

The existing agent-building tools trade power (i.e., the assistance given to the developer) against generality (i.e., their domain of applicability), covering a large spectrum. At the power end of the spectrum are the tools customized for a problem-solving method and a particular domain, such as SALT with its propose-and-revise method for design (Marcus and McDermott 1989). At the generality end are the tools applicable to a wide range of tasks and domains, such as SOAR (Jones et al. 1999) and CLIPS (Giarratano and Riley 1994). In between are the tools that are method-specific and domain independent (Chandrasekaran and Johnson 1993).

The first architectural design principle attempts to find the best trade-off between generality and power, and is illustrated by the Disciple agent architecture presented in Figure 1.

Design principle 1 (generality-power tradeoff): *Structure the architecture of the agent building tool into a reusable domain-independent learning agent shell (which ensures the generality of the tool) and domain-specific modules (which need to be built for a specific agent, to ensure the efficiency and domain-adaptability of the tool).*

Disciple-RKF is a learning agent shell (Tecuci, 1998). Its main components, presented in the left hand side of Figure 1, are:

- A problem solving component based on task reduction. It includes a modeling agent that helps the user to express his/her contributions to the problem solving process, a mixed-initiative (step-by-step) problem solving agent, and an autonomous problem solving agent.
- A learning component for acquiring and refining the knowledge of the agent, allowing a wide range of operations, from ontology import and user definition of knowledge base elements (through the use of editors and browsers), to ontology learning and rule learning.
- A knowledge base manager which controls the access and the updates to the knowledge base. Each module of Disciple can access the knowledge base only through the functions of the knowledge base manager.
- A window-based, domain-independent, graphical user interface.

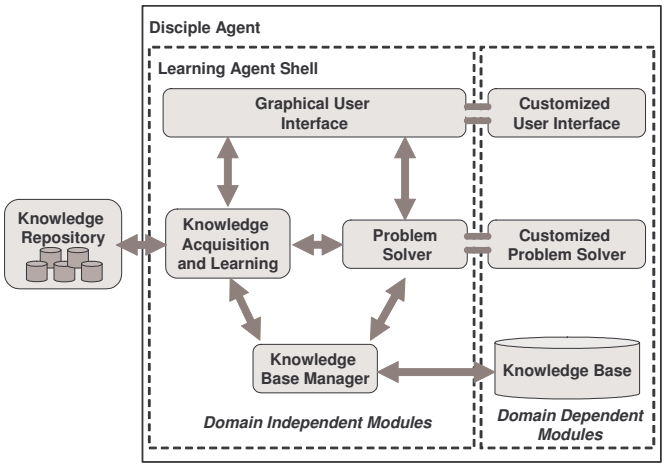

Figure 1: General architecture of a Disciple agent.

The three components in the right hand side of Figure 1 are the typical domain dependent components of a Disciple agent that was customized for a specific application:

- A customized problem solving component that extends the basic task-reduction component in order to satisfy the specific problem solving requirements of the application domain.
- Customized graphical user interfaces which are built for the specific Disciple agent to allow the SMEs and the end users to communicate with the agent as close to the way they are used to communicate as possible.
- The knowledge base of the Disciple agent.

The next design principle illustrated by the architecture of a Disciple agent is well-represented in several cognitive architectures:

Design principle 2 (cognitive functions separation): *Structure the architecture of the agent into separate modules, each dedicated to a different cognitive function, such as communication, problem solving, learning, and memory.*

Finally, the third architectural principle illustrated by Disciple-RKF is:

Design principle 3 (cognitive module as collaborative agent): *Implement each cognitive module as a collaborative agent, in a mixed-initiative framework.*

A discussion of mixed-initiative reasoning in Disciple is provided in (Tecuci et al. 2003; Boicu et al. 2003).

The next sections discuss additional design principles that are illustrated by the individual modules of Disciple-RKF.

Problem Solving

Design principle 4 (general problem solving paradigm): *Base the problem solver of the learning agent on a general problem solving paradigm (such as task reduction, state-space search, or case-based reasoning), that can be applied to a wide range of application domains, but develop a modeling framework specialized for that paradigm, to help the SMEs express their reasoning process and teach the agent.*

The concept of a learning agent shell (illustrated by Disciple-RKF), with its domain-independent problem solving and learning components (see Figure 1), is an extension of the concept of expert system shell (Clancey 1984). An expert system shell consists of a general inference engine for a given expertise domain (such as diagnosis, design, planning, scheduling, monitoring, or interpretation), and a representation formalism for encoding the knowledge base for a particular application in that domain. The idea of the expert system shell emerged from 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 system.

The problem solving component of the Disciple learning agent shell is based on the task reduction paradigm, and is more general than even the inference engine of an expert system shell. The principle of the task reduction paradigm is that a complex problem solving task (e.g. T_0 in Figure 2) is successively reduced to simpler tasks, then the solutions of the simplest tasks are found (e.g. S_{11}, … , S_{1n}) and these solutions are successively combined into the solution of the initial task (e.g. S_0), as illustrated in the left hand side of Figure 2 (Nilsson 1971; Powell and Schmidt 1988).

In the Disciple approach we have refined this general strategy to serve both as a problem solving method for the agent and as a modeling language for the SME (Bowman 2002). We did this by introducing questions and answers that guide the task reduction process, as illustrated in the right hand side of Figure 2. Finding a solution of a problem solving task (T_0) becomes an iterative process where, at each step, the expert (and/or the agent) looks for some relevant information by asking a question Q. The answer A identifies that piece of information and leads to the reduction of the current task to simpler tasks. Alternative questions correspond to alternative problem solving strategies. Several answers (e.g. A_{11a} or A_{11m}) correspond to several solutions.

Figures 3 illustrate this process with an example from the center of gravity analysis domain. In this example, the SME shows the agent how to reason to determine the centers of gravity of the opposing forces from World War II, at the time of the invasion of the island of Sicily.

Our claim is that this formulation of the task reduction paradigm is appropriate for naturally modeling a wide range of expertise domains. This claim is supported by the development of the following complex Disciple agents, as part of the DARPA's High Performance Knowledge Bases (1997-2000), and Rapid Knowledge Formation (2000-2004) programs (Tecuci and Boicu 2002a):

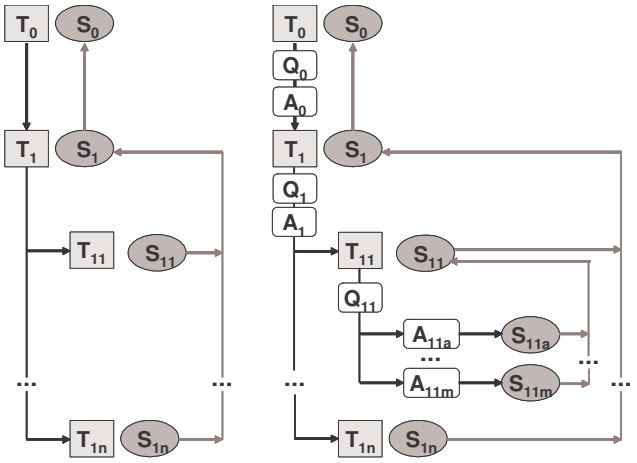

Figure 2: Question-answering based task reduction.

- A planning agent, for determining how a convoy of military vehicles can circumvent or overcome obstacles in their path, such as damaged bridges (Tecuci et al., 1999).
- A critiquing agent, for identifying strengths and weaknesses in a military course of action, based on the principles of war and the tenets of Army operation (Tecuci et al., 2001).
- An analysis agent, for identification and testing of strategic center of gravity candidates in military conflicts (Tecuci et al., 2002b).

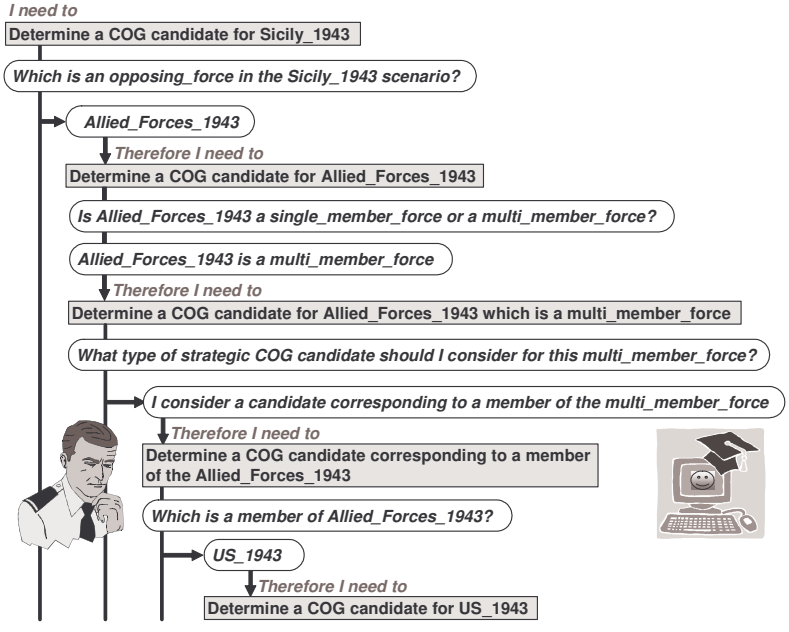

Figure 3: An illustration of task reduction.

Knowledge Base = Ontology + Rules

Design principle 5 (knowledge base structuring): *Structure the knowledge base into its more general and reusable component (the object ontology), and its more specific component (the reasoning methods or rules).*

The object ontology is a hierarchical representation of the objects and types of objects from a particular domain, such as military or medicine. That is, it represents the different kinds of objects, the properties of each object, and the relationships between objects. Figure 4, for instance, presents a fragment of the Disciple ontology for the center of gravity analysis domain (Stanescu et al., 2003).

The object ontology provides a representation vocabulary for the reduction and composition rules. Each reduction rule is an IF-THEN structure that expresses the conditions under which a task T_1 can be reduced to the simpler tasks T_{11}, … , T_{1n}. Similarly, a composition rule is an IF-THEN structure that expresses the conditions under which the solutions S_{11}, … , S_{1n} of the tasks T_{11}, … , T_{1n} can be combined into a solution S_1 of T_1 (see Figure 2). The bottom part of Figure 6 shows the informal and the formal structures of a task reduction rule.

This organization of the knowledge base is very important because it clearly separates the most general part of it (the object ontology), from its most specific part (the rules). Indeed, an object ontology is characteristic to an entire domain. In the military domain, for instance, the object ontology will include descriptions of military forces. These descriptions are most likely needed in almost any specific military application. Because building the object ontology is a very complex process, it makes sense to reuse these descriptions when developing a knowledge base for another military application, rather than starting from scratch. This justifies why the Disciple learning agent shell includes modules for ontology import and ontology merging (Barbulescu et al. 2003).

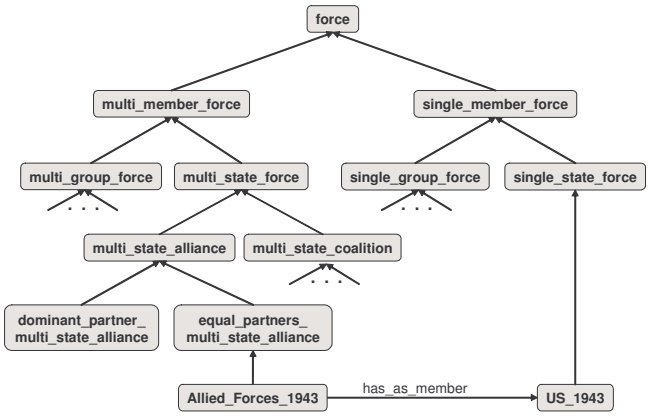

Figure 4: Ontology fragment for COG analysis.

The rules from the knowledge base are much more specific than the object ontology. Consider, for instance, two agents in the military domain, one that critiques courses of action with respect to the principles of war (Tecuci et al., 2001), and another that analyzes the center of gravity candidates of a force (Tecuci et al., 2002). While both agents need to reason with military forces, their reasoning rules are very different, being specific not only to their particular application (course of action critiquing vs. center of gravity analysis), but also to the SMEs whose expertise they encode.

Design principle 6 (partially learned knowledge): *Design the agent to allow the representation, use, and continuous refinement of partially learned knowledge.*

The Disciple agents have the capability of incrementally learning reasoning rules from the SMEs, as will be discussed in the next section. This capability, in turn, requires the capability of representing and reasoning with incompletely learned knowledge pieces. At the basis of Disciple's learnable representation of knowledge are the notion of plausible version space (Tecuci 1998) and the use of the object ontology as an incomplete generalization hierarchy for learning. A plausible version space is an approximate representation for a partially learned concept, as illustrated in Figure 5. The partially learned concept is represented by a plausible upper bound concept which, as an approximation, is more general than the concept E_h to be learned, and by a plausible lower bound concept which, again as an approximation, is less general than E_h. During learning, the two bounds (which are first order logical expressions) converge toward one another through successive generalizations and specializations, approximating E_h better and better. Notice, however, that these generalization and specialization operations are based on an ontology that is itself evolving during knowledge acquisition and learning (for instance, by adding new concepts in the hierarchy from Figure 4). Therefore Disciple addresses the complex problem of learning in the context of an evolving representation language. All the knowledge pieces from the agent's knowledge base (objects, tasks, and rules) are

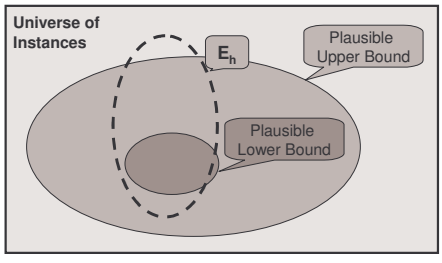

Figure 5: A representation of a plausible version space.

represented using plausible version spaces, and may be learned by the agent from specific examples provided by the SME. For instance, the rule from the bottom of Figure 6 was learned from the example shown at the top of Figure 6, and the explanation of this example. Notice the plausible version space condition of the formal structure of the rule, represented by a plausible upper bound condition and a plausible lower bound condition.

Learning

The main strength of a Disciple agent is its ability to learn from an SME. This capability is based on three design principles that will be briefly described in the following.

Design principle 7 (integrated problem solving and learning): *Design the agent to support integrated problem solving and learning, where the SME and the agent solve problems in cooperation and the agent learns from the problem solving contributions of the SME, and from its own problem solving attempts.*

This form of situated learning during problem solving facilitates the definition of examples by the SME, and the learning and refinement of the rules by the agent.

Initially, the interaction between the SME and a Disciple agent has the flavor of a teacher-student interaction, where the SME shows the agent each problem solving step required to solve a task, and the agent learns a general task reduction rule from each step. For instance, the rule from the bottom of Figure 6 was learned from the SME-provided example shown at the top of Figure 6 (which was actually provided as the bottom step from the problem solving tree in Figure 3).

As Disciple learns new rules from the SME, the interaction between the SME and Disciple evolves from a teacher-student interaction, toward an interaction where both collaborate in solving a problem. During this mixed-initiative problem solving phase, Disciple learns not only from the contributions of the SME, but also from its own successful or unsuccessful problem solving attempts, which lead to the refinement of the rule's plausible version space condition (Boicu et al., 2000).

Design principle 8 (integrated teaching and learning): *Design the agent to support integrated teaching and learning, where the agent helps the SME to teach it (e.g. by asking relevant questions), and the SME helps the agent to learn (e.g. by providing examples, hints and explanations).*

Let us consider, for instance, the example from the top of Figure 6, which was provided by the SME. The SME helps the agent to understand that the meaning of the Question–Answer pair from this example is "Allied_Forces_1943 has_as_member US_1943", and that this is the reason of performing the reduction represented by this example. As a consequence, Disciple automatically generates the plausible version space rule from the bottom of Figure 6, where the plausible lower bound condition is the minimal generalization of the above explanation piece and the other objects from the example, and the plausible upper bound is the maximal generalization. Both these generalizations are based on the agent's ontology, a fragment of which is shown in Figure 4. The middle part of Figure 6 shows also the informal structure of the learned rule. This informal structure preserves the natural language of the SME and is used in agent-user communication. The formal structure from the bottom part of Figure 6 is used in the actual reasoning of the agent.

Design principle 9 (multistrategy learning): *Design the learning module of the agent as a multistrategy learner that synergistically integrates several learning strategies, taking advantage of their complementary strengths to compensate for each other's weaknesses* (Michalski and Tecuci, 1994).

For instance, Figure 7 represents the rule refinement method of Disciple that integrates learning by analogy and experimentation, learning from examples, and learning

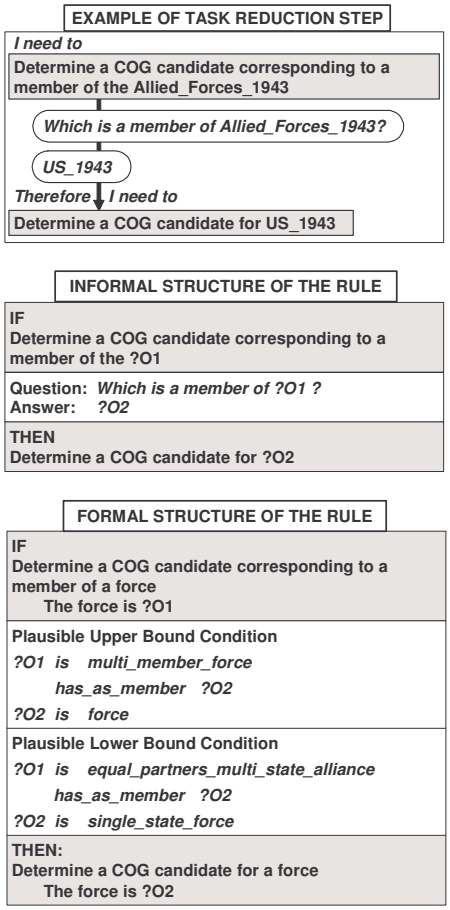

Figure 6: An example and the rule learned from it.

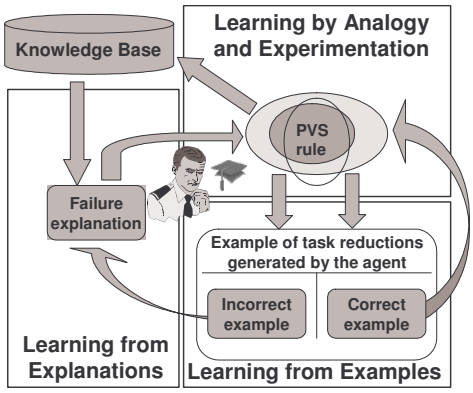

Figure 7: The rule refinement method of Disciple.

from explanations. The plausible upper bound condition of a learned rule allows it to apply to situations that are analogous with the one from which the rule was learned. If the SME judges this application as correct, then this represents a new positive example of the rule, and the plausible lower bound condition is generalized to cover it. Otherwise, the agent will interact with the SME to find an explanation of why the application is incorrect, and will specialize the rule's conditions appropriately. Rule refinement could lead to a complex task reduction rule, with additional except-when conditions which should not be satisfied in order for the rule to be applicable (Boicu et al. 2000).

Agent Development Methodology

As indicated in Figure 8, the Disciple approach covers all the phases of agent development and use. First, the knowledge engineer works with the SME to develop an initial model of how the SME solves problems, based on the task reduction paradigm (1). The model identifies also the object concepts that need to be present in Disciple's ontology so that it can perform this type of reasoning. These object concepts represent a specification of the needed ontology, specification that guides the process of importing ontological knowledge from existing knowledge repositories (2). The knowledge engineer and the SME extend the imported ontology and define the scripts for scenario elicitation (3).

The result of these initial knowledge base development steps is a complete enough object ontology that will allow a SME to train the Disciple agent how to solve problems, with limited assistance from a knowledge engineer. First, during scenario elicitation (4), Disciple guides the SME to describe a scenario and then creates a formal description of it consisting of instances in the object ontology. Then, using the initial domain model as a guide, the SME expresses in English how he or she solves specific problems through task reduction (5). During task learning (6) the SME and Disciple collaborate in formalizing the tasks from the SME's examples, and Disciple learns general task patterns. Then the SME helps Disciple to understand each task reduction step and Disciple learns general task reduction rules (7). Disciple uses the partially learned rules in problem solving and then refines them based on SME's feedback (8). At the same time, Disciple extends the object ontology with new objects and features (9). During problem solving (10) Disciple automatically solves problems, providing solutions and detailed justifications.

While the above is the normal sequence of these steps, there is also the need to return to a previous step. For instance, while performing the step-by-step problem solving and rule refinement, the SME may need to define a new reduction which requires him/her to perform modeling, task learning and rule learning.

It is possible for copies of Disciple agents to be trained in parallel by different SMEs. In this case the individual knowledge bases have to be merged into an integrated knowledge base (11). The resulting knowledge base can also be exported for use in the development of new agents (12). The last phase represents the use of the developed Disciple agent by the intended end-users (13).

Evaluation

Over the last four years, several experiments have been performed at the US Army War College, with increasingly more capable versions of Disciple.

During three successive sessions of the "589jw Military Applications of Artificial Intelligence" course (held in Spring 2001, Spring 2002, and Spring 2003, respectively), a total of 38 US and international officers from all branches of the military have trained personal Disciple agents, with limited assistance from knowledge engineers. At the end of each experiment, we have asked the military experts to express their disagreement or agreement with the following statement: *"I think that a subject matter expert can use Disciple to build an agent, with limited assistance from a knowledge engineer."*

Out of the 38 experts, 10 strongly agreed with the above statement, 20 of them agreed, 7 were neutral, and 1 disagreed, which represents a very encouraging result.

Successive versions of trained Disciple agents were also used in 8 sessions of the "319jw Case Studies in COG Analysis" course (the COG course), by a total of 71 military students. In the COG courses, the students have used Disciple as an intelligent assistant that helped them to develop a Center of Gravity analysis of a war scenario. For instance, all the 8 students from the Spring 2003 session of the COG course have agreed or strongly

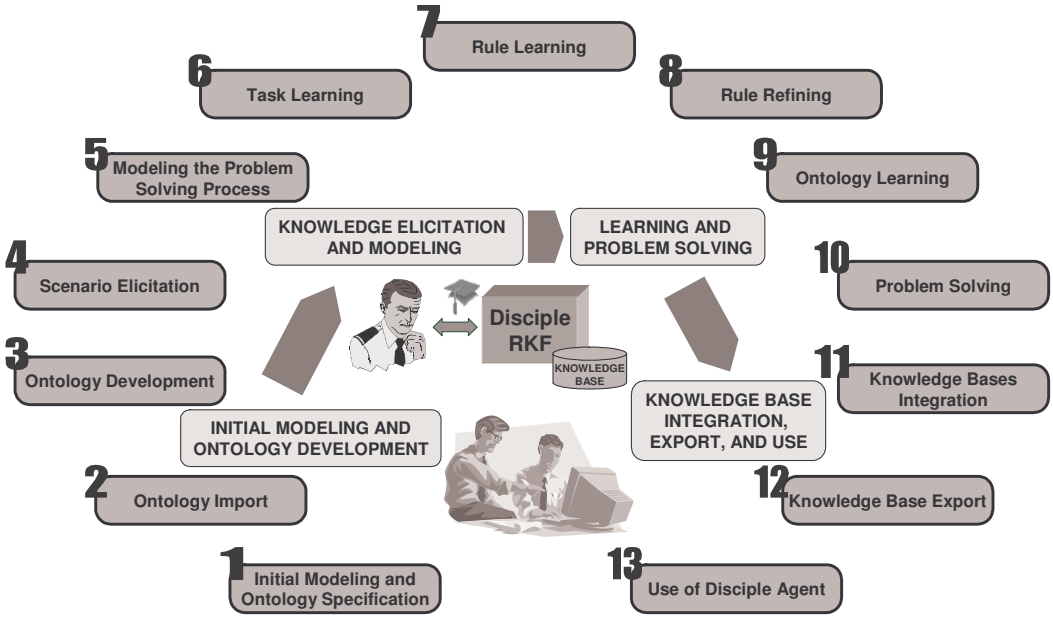

Figure 8: The phases of agent development and use.

agreed with the following statements: *"The use of Disciple is an assignment that is well suited to the course's learning objectives"* and *"Disciple should be used in future versions of this course"*.

The significance of this result is that it shows that the Disciple approach can be used to build practical agents for complex real-world problems.

More evaluation results are presented in (Tecuci et al., 2001; Tecuci et al., 2002).

While several of the design principles discussed in this paper are not unique to Disciple, what is unique is the integrated application of all of them within a single agent architecture, which proved to be very successful, as suggested by the above evaluation results.

Acknowledgments

This research was sponsored by DARPA, AFRL, AFMC, USAF, under agreement number F30602-00-2-0546, by the AFOSR under grant no. F49620-00-1-0072, and by the US Army War College. Several persons supported this effort, including David Gunning, Murray Burke, William Rzepka, Jerome Comello, William Cleckner, Douglass Campbell, David Brooks, David Cammons, and Michael Bowman.

References

Barbulescu, M., Balan, G., Boicu, M., Tecuci, G. 2003. Rapid development of large knowledge bases. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics,* 3: 2169-2174, Institute of Electrical and Electronics Engineers, Inc.

Boicu, M. 2002. Modeling and Learning with Incomplete Knowledge, Ph.D. diss., Dept. of Computer Science, George Mason Univ.

Boicu M., Tecuci G., Marcu D., Bowman M., Shyr P., Ciucu F., and Levcovici C. 2000. Disciple-COA: From Agent Programming to Agent Teaching. In *Proceedings of the Seventeenth International Conference on Machine Learning,* Stanford, CA: Morgan Kaufmann.

Boicu M., Tecuci G., Marcu D., Boicu C., and Stanescu B. 2003. Mixed-Initiative Control for Teaching and Learning in Disciple. In *Proceedings of IJCAI-2003 Workshop on Mixed-Initiative Intelligent Systems*, 9-16, Menlo Park, CA: AAAI Press.

Bowman, M. 2002. A Methodology for Modeling Expert Knowledge that Supports Teaching Based Development of Agents. Ph.D. diss., Dept. of Computer Science, George Mason Univ.

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

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,* 239-280, Springer-Verlag.

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*, 361-381. Reading, MA: Addison-Wesley.

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

Jones, R. M., Laird, J. E., Nielsen P. E., Coulter, K., Kenny, P., and Koss, F. 1999. Automated Intelligent Pilots for Combat Flight Simulation, *AI Magazine 20(1)*: 27-42.

Marcus S. and McDermott J. 1989. SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems, *Artificial Intelligence* 39: 1-37.

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

Nilsson, N.J. 1971. *Problem Solving Methods in Artificial Intelligence*. New York, NY: McGraw-Hill.

Powell G.M. and Schmidt C.F. 1988. A First-order Computational Model of Human Operational Planning, *CECOM-TR-01-8,* US Army CECOM, Fort Monmouth, New Jersey.

Stanescu B., Boicu C., Balan G., Barbulescu M., Boicu M., Tecuci G. 2003. Ontologies for Learning Agents: Problems, Solutions and Directions. In *Proceedings of the IJCAI-03 Workshop on Workshop on Ontologies and Distributed Systems,* 75-82. Acapulco, Mexico, AAAI Press, Menlo Park, CA.

Tecuci G. 1988. DISCIPLE: A Theory, Methodology and System for Learning Expert Knowledge, Thèse de Docteur en Science, University of Paris-South.

Tecuci, G. 1998. *Building Intelligent Agents: An Apprenticeship Multistrategy Learning Theory, Methodology, Tool and Case Studies*. London: 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. *Proc. AAAI/IAAI,* pp. 250-257, AAAI Press, Menlo Park CA.

Tecuci G., Boicu M., Bowman M., and Marcu D., with a commentary by Burke M. 2001. An Innovative Application from the DARPA Knowledge Bases Programs: Rapid Development of a High Performance Knowledge Base for Course of Action Critiquing, *AI Magazine,* Vol. 22, No. 2, pp. 43-61.

Tecuci G., Boicu M. 2002a. Military Applications of the Disciple Learning Agent. In Jain L, ed., *Advanced Information Systems in Defense and Related Applications,* 337-376. Springer Verlag.

Tecuci G., Boicu, M., Marcu, D., Stanescu, B., Boicu, C., and Comello, J. 2002b. Training and Using Disciple Agents: A Case Study in the Military Center of Gravity Analysis Domain. *AI Magazine 23(4)*: 51–68.

Tecuci G., Boicu M., Marcu D. 2003. Toward a Disciple-based Mixed-Initiative Cognitive Assistant. In *Proceedings of IJCAI-2003 Workshop on Mixed-Initiative Intelligent Systems,* 131-137. Menlo Park, CA: AAAI Press.