An Integrated Shell and Methodology for Rapid Development of Knowledge-Based Agents

Gheorghe Tecuci, Mihai Boicu, Kathryn Wright, Seok Won Lee, Dorin Marcu, and Michael Bowman

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

Abstract

This paper introduces the concept of *learning agent shell* as a new class of tools for rapid development of practical end-to-end knowledge-based agents, by domain experts, with limited assistance from knowledge engineers. 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. The paper describes a specific learning agent shell and its associated agent building methodology. The process of developing an agent 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. The shell and methodology represent a practical integration of knowledge representation, knowledge acquisition, learning and problem solving. This work is illustrated with an example of developing a hierarchical non-linear planning agent.

Introduction

This paper describes recent progress in developing an integrated shell and methodology for rapid development of practical end-to-end knowledge-based agents, by domain experts, with limited assistance from knowledge engineers.

We use the term "knowledge-based agent" to broadly denote a knowledge based system that interacts with a subject matter expert (SME) to learn from the expert how to assist him or her with various tasks.

Our work advances the efforts of developing methods, tools, and methodologies for more rapidly building knowledge-based systems. One of the early major accomplishments of these efforts was 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, 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, and the goal of reusing the inference engine for a new system.

Currently, we witness a similar separation at the level of the knowledge base, which is more and more regarded as consisting of two main components: an ontology that defines the concepts from the application domain, and a set of problem solving rules (methods) 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 (e.g. rules for an agent that assists a military commander in critiquing courses of action, or rules for an agent that assists in planning the repair of damaged bridges or roads).

The emergence of domain ontologies is primarily a result of terminological standardization in more and more domains to facilitate automatic processing of information, particularly information retrieval. Some of existing 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 system. However, sharing and reusing the components of different knowledge representation systems are hard research problems because of the incompatibilities in their implicit knowledge models. Recently, the Open Knowledge Base Connectivity (OKBC) protocol has been developed as a standard for accessing knowledge bases stored in different frame representation systems (Chaudhri et al. 1998). OKBC 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.

The existence of domain ontologies facilitates the process of building the knowledge base, which may reduce to one of reusing an existing ontology and defining the application specific problem solving rules. This effort, however, should not be underestimated. Several decades of knowledge engineering attests that the traditional process by which a knowledge engineer interacts with a domain expert to manually encode his or her knowledge into rules is long, difficult and error-prone. Also, automatic learning of rules from data does not yet provide a practical solution to this problem. An alternative approach to acquiring problem solving rules is presented in (Tecuci 1998). In this approach an expert interacts directly with the agent to teach it 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. During the interaction with the expert the agent learns problem solving rules by integrating a wide range of knowledge acquisition and machine learning techniques, such as apprenticeship learning, empirical inductive learning from examples and explanations, analogical

learning and others.

Based on these developments and observations, we propose the concept of "learning agent shell" as a tool for building intelligent agents by domain experts, with limited assistance from knowledge engineers. A learning agent shell consists of a learning and knowledge acquisition engine and an inference engine that support building an agent with a knowledge base composed of an ontology and a set of problem solving rules.

In this paper we present the Disciple Learning Agent Shell (Disciple-LAS) and its methodology for rapid development of knowledge based agents, which relies upon importing ontologies from existing repositories using the OKBC protocol and on teaching the agents to perform various tasks through cooperative problem solving and apprenticeship multistrategy learning. Among the major developments of Disciple-LAS with respect to previous versions of Disciple, we could mention:

- the adoption of the OKBC knowledge model as the basic representation of Disciple's ontology and the extension of the Disciple's apprenticeship multistrategy learning methods to deal with this more powerful knowledge model. These methods have become more knowledge-intensive and less dependent on expert's help, especially through the use of more powerful forms of analogical reasoning. The primary motivation of this extension was to facilitate the ontology import process.
- the development and integration into Disciple of a general purpose cooperative problem solver, based on task reduction. It can run both in a step by step mode and in autonomous mode.
- the development of an integrated methodology for building end-to-end agents.

With respect to the Disciple-LAS shell and methodology we formulate 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.

The rest of the paper is organized as follows. We first introduce the Disciple modeling of an application domain. Then we present the architecture of Disciple-LAS, the specification of an agent built with Disciple, and the agent building methodology. We present experimental results of building the specified agent, and we conclude the paper.

Domain Modeling for Integrated Knowledge Acquisition, Learning and Problem Solving

We claim that 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.

As problem solving approach we have adopted the classical task reduction paradigm. 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:

1. Objects that represent either specific individuals or sets of individuals in the application domain. The objects are hierarchically organized according to the generalization relation.

2. Features and sets of features that are used to further describe objects, other features and tasks. Two important features 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, and are also hierarchically organized.

3. Tasks and sets of tasks that are hierarchically organized. A task is a representation of anything that the agent may be asked to accomplish.

The objects, features and tasks are represented as frames, according to the OKBC knowledge model, with some extensions.

4. Examples of task reductions, such as:

TR: If the task to accomplish is T_1
then accomplish the tasks $T_{11}, \dots, T_{1n}$

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.

5. Explanations of task reduction examples. An explanation 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 to a specific task reduction:

the task reduction TR is correct because E

One could more formally represent the relationship between TR and E as follows:

$E \rightarrow TR$, or $E \rightarrow (accomplish(T_1) \rightarrow accomplish(T_{11}, \dots, T_{1n}))$

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

However, the example and its explanation can also be represented in the equivalent form:

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

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

If the task to accomplish is T_1 (1)
and E holds
then accomplish the tasks $T_{11}, \dots, T_{1n}$

6. Rules. The rules are generalizations of specific reductions, such as (1), and are learned by the agent through an interaction with the domain expert, as described in (Tecuci, 1998):

If the task to accomplish is T_{1g} and (2)
E_h holds
then accomplish the tasks $T_{11g}, \dots, T_{1ng}$

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. An except-when condition is a generalization of the explanation of why a negative example of a rule does not represent a correct task reduction. Finally, the rule may also have "except-for" conditions (that specify instances that are negative exceptions of the rule) and "for" conditions (that specify positive exceptions).

The ontology of objects, features and tasks serves as the generalization hierarchy for Disciple-LAS. An example is basically generalized by replacing its objects with more general objects from the ontology. In the current version of Disciple-LAS the features and the tasks are not generalized, but they are used for analogical reasoning and learning.

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 could 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 (3)
PUB: E_g holds
PLB: E_s holds
then accomplish the tasks T_{11g}, ... , 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 potentially be applied for tasks that are similar to T_{1g}. In such a case the reductions would be 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, that are used by the agent to further improve the rule.

Architecture of the Disciple-LAS

The architecture of Disciple-LAS is presented in Figure 1. 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, or for importing knowledge from KIF files (Genesereth and Fikes, 1992).
- 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.

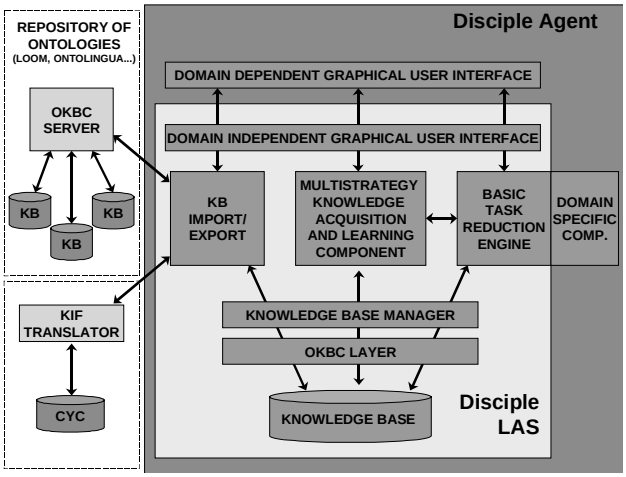

Figure 1: General architecture of the Disciple-LAS

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 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.

Disciple-LAS is implemented in JAVA and LISP, in a client-server architecture that assures portability, multi-user development of agents, and fast (socket) connection.

Rapid Development of a Workaround Agent

The integrated Disciple-LAS and methodology were developed as part of the DARPA's High Performance Knowledge Bases Program (Cohen et al. 1998), and were applied to rapidly build a planning agent for solving the workaround challenge problem. We will use this problem to illustrate the Disciple methodology. The problem consisted of assessing how rapidly and by what method a

military unit can reconstitute or bypass damage to an infrastructure, such as a damaged bridge or a cratered road (Alphatech 1998).

The input to the agent includes two elements: (1) 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's banks, the river's speed, depth and width), (2) 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 workaround 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: (1) 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; (2) a partially ordered plan of engineering actions to perform the repair, and the minimum as well as the expected time that each of these actions require; and (3) a set of required resources for the entire plan and for each action.

Workaround generation requires detailed knowledge about the capabilities of various types of engineering equipment and about their 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 acquired from military experts and Army field manuals.

The Methodology for Building Agents

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

1. Specification of the problem

The SME and the knowledge engineer generally accomplish this step. In the HPKB program, the workaround challenge problem was defined in a 161-page report created by Alphatech (1998). This report already identified many concepts needed to be represented in 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.

2.Modeling 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 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, and they could be applied in conjunction with other types of problem solvers, this being one of our future research directions.

In the case of the workaround challenge problem, task reduction proved to actually be a very natural way to model it, the problem solver being a hierarchical non-linear planner.

During the modeling process, the domain is partitioned into classes of typical problem solving scenarios, and for each such scenario, an informal task reduction tree is defined. Examples of problem solving scenarios for the workaround domain are: workaround a damaged bridge by performing minor preparation to install a fixed bridge over the river, workaround a damaged bridge by performing gap reduction to install a fixed bridge, workaround a damaged bridge through fording, workaround a gap in the bridge by using a fixed bridge, workaround a damaged bridge by installing a ribbon bridge, etc.

There are several important results of the modeling process: (1) an informal description of the agent's tasks is produced, (2) additional necessary concepts and features are identified, (3) conceptual task reduction trees are produced that will guide the training of the agent by the domain expert.

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. Also, an interface for displaying maps with the damaged area was integrated into the agent architecture.

4. Importing concepts and features from other ontologies

As a result of the first two steps of the methodology, a significant number of necessary concepts and features have been identified. Interacting with the Knowledge Import/Export Module, the domain expert and the knowledge engineer attempt 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 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.

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 military load classes). The extent of knowledge import was more limited than it could have been because the LOOM's ontology was developed at the same time as that of Disciple, and we had to define concepts that have later been also 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.

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 the tasks informally specified in step 3. New tasks, objects and features, could also be defined during the next step of training the agent.

6. Training the agent for its domain-specific tasks

While the previous steps are more or less standard in any agent building methodology (with the possible exception of the agent customization step and the ontology importing step), the training of the agent is a characteristic step of the Disciple agent building methodology.

The main result of this step is a set of problem solving rules. Defining correct problem solving rules in a traditional knowledge engineering approach is known to be very difficult. The process implemented in Disciple is based on the following assumptions:

- it is easier for an SME to provide specific examples of problem solving episodes than general rules;
- it is easier for an SME to understand a phrase in agent's language (such as an example or an explanation) than to create it;
- it is easier for an SME to specify hints on how to solve a problem than to give detailed explanations;
- it is easier for the agent to assist the SME in the knowledge acquisition process if the agent has more knowledge.

As a consequence, Disciple incorporates a suite of methods that reduce knowledge acquisition and learning from an SME to the above simpler operations, and are based on increasing assistance from the agent. These methods include:

- methods to facilitate the definition of examples of task reductions;
- heuristic, hint-based and analogy-based methods to generate the explanations of a task reduction;
- analogy-based method to generalize examples to rules;
- methods to generate relevant examples to refine the reduction rules, etc.

During this step, the expert teaches Disciple to solve problems in a cooperative, step by step, problem solving scenario. 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 again none of these rules apply, then the agent may attempt to use rules corresponding to tasks known to be similar with the one to be reduced. For instance, to reduce the task "Workaround a destroyed bridge using a floating bridge with slope reduction", the agent may consider the reduction rules corresponding to the similar task "Workaround a destroyed bridge using a fixed bridge with slope reduction."

If the solution was defined or modified by the expert, then it represents an initial example for learning a new reduction rule. To learn the rule, the agent will first try to find an explanation of why the reduction is correct. Then the example and the explanation are generalized to a plausible version space rule. The agent will attempt various strategies to propose plausible explanations from which the user will choose the correct ones. The strategies are based on an ordered set of heuristics. 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 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. For instance, if the task is to workaround a damaged bridge using a fixed bridge over the river gap, then the decision of whether to employ (or, equivalently, the explanation of why to employ) an installation of the bridge with minor preparation of the area, or with gap reduction, or with slope reduction, depends upon the specific relationships between the dimensions of the bridge and the dimensions of the river gap. The goal is to have the agent propose explanations ordered by their plausibility and the expert to choose the right ones, rather than requiring the explanations from the expert.

This above strategy works well when the agent already has a significant amount of knowledge related to the current reduction. 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 reduction of the task "Workaround damaged bridge using an AVLB70 bridge over the river gap", to the task "Install AVLB70 with gap reduction over the river gap". The expert can give the agent a very general hint, such as, "Look for correlations between the river gap and AVLB70." A more specific hint would be "Look for correlations between the length of the river gap and the lengths of the gaps breachable with AVLB70." Such hints will guide the agent in proposing the correct explanation: "The length of the river gap is greater than the length of AVLB70, but less than the maximum gap that can be reduced in order to use AVLB70".

The goal is to allow the expert to provide hints or incomplete explanations rather than detailed explanations.

The above situations occur when the expert provides the reduction of the current task and will ultimately result in learning a new task reduction rule.

We will now briefly consider some of the other possible cases, where the agent proposes reductions based on the existing rules. If the reduction was accepted by the expert 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. This explanation will be used to refine rule's 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 also be 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 used to guide the learning of new rules for MGB and Bailey. These rules have in turn facilitated the learning of new rules for floating bridges. The rules for floating bridges have facilitated the learning of the rules for ribbon rafts, and so on.

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 the error and to help the agent to fix it, as described before.

The developed agent can be used by a non-expert user. More interesting is, however, the case where the agent continues to act as an assistant to the expert, solving problems in cooperation, continuously learning from the expert, and becoming more and more useful.

In the case of the workaround domain, the evaluator 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.

As has been shown above, the Disciple-LAS shell and methodology provide solutions to some of the issues that have been found to be limiting factors in developing knowledge-based agents:

- 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.

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 (Cohen et al. 1998). The evaluation consisted of two phases, each comprising a test and a re-test. In the first phase, the systems were tested on 20 problems 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 the best 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 human expert. Bonus points were awarded when systems gave better answers than the expert and these answers were used as standard 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 succeeded to enter the evaluation with a system that had a fully developed KB. The other three teams entered the evaluation with systems that had incompletely developed knowledge bases. Figure 2 shows a plot of the overall coverage of each system against the overall correctness of that system, for each of the two phases of the evaluation.

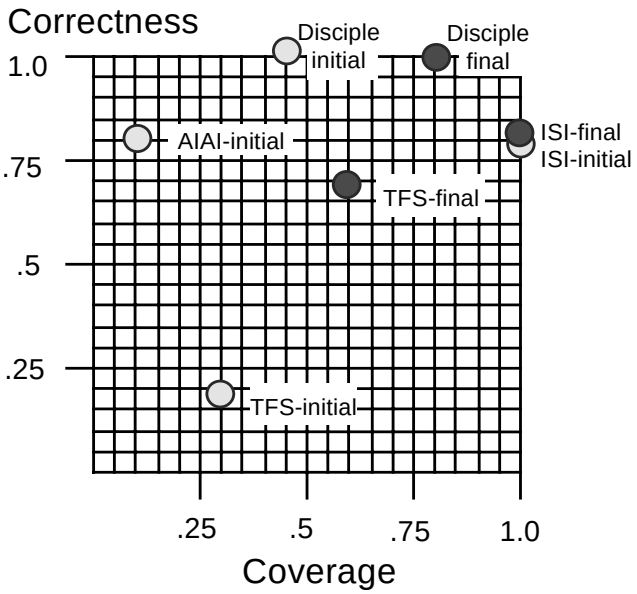

Figure 2: Evaluation results.

We entered the evaluation with a workaround agent the knowledge base of which was covering 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 3. 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, within its scope, the Disciple agent 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 expert. 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 similar way with 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 on others fixed bridges were emplaced. These solutions were adopted by the expert and used as standard for improving all the systems. For this reason, although the agent also made some mistakes, the overall correctness of its solutions was practically as high as that of the expert's solutions. This result supports the 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.

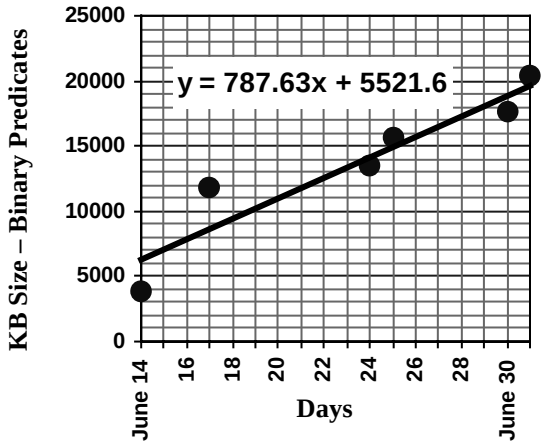

Figure 3: KB Development time.

Finally, our workaround generator had also a very good performance, being able to generate a solution in about 0.3 seconds, on a medium power PC. This supports the 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 the promising new technologies.

Related Work

From the point of view of the methods and techniques employed, this work is mostly related to the work on apprenticeship learning that has produced experimental agents that assimilate new knowledge by observing and analyzing the problem solving steps contributed by their expert users through their normal use of the agents (Mahadevan et al. 1993, Wilkins 1993). Disciple-LAS is different from these agents in terms of the types of learning employed. Also it has been scaled up and developed into a general integrated shell and methodology for building practical end-to-end agents.

Disciple-LAS is also related to the tools for building knowledge-based systems. Many of these tools provide an inference engine, a representation formalism in which the KB 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).

With respect to the power-generality trade-off, Disciple-LAS takes a different approach. It provides a set of general and powerful modules for knowledge acquisition and learning that are domain-independent and are incorporated as such in a developed agent. However, for the interface and the problem solver, the Disciple shell contains a generic graphical-user interface and a problem solver based on task-reduction. Therefore, for a given application domain, one has to develop additional, domain-specific interfaces and to further develop the problem solver, in order to create an easy to train and a useful agent. In spite of its generality, and due to its powerful learning capabilities, Disciple's support in knowledge acquisition is similar to that of the specific tools. Moreover, it provides support in all the stages of knowledge base construction, both ontology and rules creation, and their refinement. Many of the other systems stress either initial knowledge creation, or its refinement. Finally, most of the other tools are intended for the knowledge engineer, while Disciple-LAS is oriented toward direct knowledge acquisition from a human expert, attempting to limit as much as possible the assistance needed from the knowledge engineer.

As compared with Disciple-LAS, the other tools used in the HPKB project to solve the workaround challenge 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.

Conclusion and Future Research

The main result of this paper is an integration of knowledge representation, knowledge acquisition, learning and problem solving into an agent shell and methodology for efficient development of practical end-to-end knowledge-based agents, by domain experts, with limited assistance from knowledge engineers. This approach is based on the reuse and adaptation of previously developed knowledge, and on a natural interaction with the domain expert which are achieved through 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. Finally, there is the synergism between the expert and the agent in solving a problem, where the agent solves the more routine parts of the problem and the expert solves the more creative parts. In the process, the agent learns from the expert, gradually evolving toward an intelligent agent.

There are, however, several weaknesses of this approach that we plan to address in the future. For instance, the initial modeling of the domain, which is critical to the successful development of the agent, is not yet supported by the shell. 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. We also need to develop a more powerful and natural approach to hint specification by the expert. The current types of allowable hints do not constrain enough the search for explanations. Also, some of them are not very intuitive for the expert. Finally, we are investigating how the learning methods of the agent could become even more knowledge intensive, primarily through the use of more powerful methods of analogical reasoning.

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

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.

Genesereth M.R. and Fikes R.E. 1992. Knowledge Interchange Format, Version 3.0 Reference Manual. KSL-92-86, Knowledge Systems Laboratory. Stanford University.

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.

Mahadevan, S., Mitchell, T., Mostow, J., Steinberg, L., and Tadepalli, P. 1993. An Apprentice Based Approach to Knowledge Acquisition, *Artificial Intelligence*, 64(1):1-52.

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.

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

Wilkins, D. 1993. Knowledge base refinement as improving an incomplete and incorrect domain theory. In Buchanan, B. and Wilkins, D. eds. *Readings in Knowledge Acquisition and Learning*, San Mateo, CA: Morgan Kaufmann.