In Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99) Workshop on "Ontology Management", WS-99-13, pp. 7-16, July 18-19, Orlando, Florida, AAAI Press, Menlo Park, CA. 1999

A Problem-Oriented Approach to Ontology Development

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

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

Abstract

This paper presents the approach to ontology development that is part of the Disciple-LAS shell and methodology for the building of knowledge-based agents. A characteristic feature of this approach is that a detailed specification of the ontology to be developed results from a conceptual modeling of the application domain of the knowledge-based agent to be built. This specification guides the process of building the ontology which consists in importing knowledge from external knowledge servers, and in using the ontology building tools of Disciple. Knowledge import and reuse are facilitated by the fact that the representation of the ontology is based on the OKBC knowledge model. This approach is used to develop an ontology for an agent that critiques military courses of actions.

Introduction

We are developing an approach to rapid building of practical end-to-end knowledge-based agents, by subject matter experts, with limited assistance from knowledge engineers. This approach is implemented into a learning agent shell, called Disciple-LAS, which consists of a knowledge acquisition and learning engine and an inference engine. Disciple-LAS is used by a subject matter expert and a knowledge engineer to develop a specific knowledge-based agent. Central to this approach is an architectural separation of the agent's knowledge base into two main components, an ontology that defines the concepts from the application domain, and a set of problem solving rules expressed in terms of these concepts.

The process of building the knowledge base of a Disciple agent consists of three main steps:

1. A modeling step, where the expert and the knowledge engineer define at a conceptual level how the agent will perform its tasks. This step results in an informal specification of the concepts needed to be represented in the agent's ontology.
2. An ontology creation step, where the knowledge engineer and the expert import some of the concepts identified in the previous step from existing repositories of knowledge, and define the rest of the concepts. This step results in an initial knowledge base that contains an incomplete ontology and no rules.
3. A teaching step where the subject matter expert teaches the agent how to perform its tasks in a way that resembles how the expert would teach a human apprentice when solving problems in cooperation. During this step the agent will learn problem solving rules from the expert, and will also extend and update the ontology.

The developed knowledge-based agent can be used by the subject matter expert as an assistant, or it can be used by a non-expert user.

In this paper we address the ontology creation and maintenance issues that occur within the Disciple framework, and how they are dealt with in Disciple-LAS.

Disciple-LAS is currently applied to develop an agent that critiques military courses of actions (COA). We will therefore start by briefly describing this defense application of Disciple which will be used to illustrate our approach to ontology management.

Then we will describe the knowledge representation of a Disciple agent, and we will clarify the structuring of the knowledge base into an ontology and a set of problem solving rules. The main point is that Disciple uses a learning-oriented representation, where the ontology serves as the generalization hierarchy that is used to learn general rules from specific examples. Another important aspect is that the representation of the ontology is based on the OKBC knowledge model, to facilitate knowledge import from OKBC compliant knowledge servers.

Then we will describe the process of modeling an application domain. The important aspect here is that this process identifies the concepts that are needed in the ontology in order for the COA agent to perform its tasks. This is in contrast to other approaches to ontology construction, such as the one used with CYC (Lenat 1995), where the goal is to develop a general ontology that tries to abstract away from the specifics of a particular application domain. In the Disciple approach, the goal is to create a domain specific ontology that is very well suited to solving problems in the considered application domain.

Once we have identified, at an informal, conceptual level, which are the needed concepts, we can start building the ontology. We will first present our approach to importing concepts from external repositories of knowledge. This process is guided by the concepts identified during the modeling process.

Next we will briefly present the Disciple ontology creation and maintenance tools that allow one to further extend and update the ontology to include all the concepts identified as necessary during the modeling process. An important aspect is the development of tools that are dedicated to the type of user. In Disciple we distinguish between a knowledge engineer, a subject matter expert, and

an end-user. Each tool will have different characteristics, depending on the user type. For instance, a tool for the knowledge engineer will have many options and will allow its user to have total control over the operation of the tool. On the contrary, a tool for an end-user will have only a few options and will provide close guidance.

Once the ontology has been created, the subject matter expert can start teaching the Disciple agent how to perform its tasks. From its interaction with the subject matter expert the agent will learn general problem solving rules and will also update and extend the ontology. Although we consider this to be the most innovative aspect of the Disciple approach, we will only provide here a very general overview, stressing only the aspects related to the evolution of the ontology. The teaching and learning methods of Disciple are described in (Tecuci, 1998; Tecuci et al., 1999).

Finally, we will address some important design and implementation issues that help in developing evolving and scalable ontology management tools in a research environment. We will first present the design of the Knowledge Base Manager of Disciple that facilitates the extension of Disciple's knowledge representation. We will also briefly discuss the high-speed graph search algorithms used by the KB management functions, and some solutions to extend the physical size of the KB.

We will conclude the paper with some directions for future research.

COA Critiquing

Disciple-LAS and methodology are used to build a critiquing agent that critiques military courses of actions. A military COA is a preliminary outline of a plan for how a military unit might attempt to accomplish a mission.

A COA is not a complete plan in that it leaves out many details of the operation such as exact initial locations of friendly and enemy forces. After receiving orders to plan for a mission, a commander and staff complete a detailed and practiced process of analyzing the mission, conceiving and evaluating potential COAs, selection of a COA, and the preparation of detailed plans to accomplish the mission based on the selected COA.

The general practice is for the staff to generate several COAs for a mission, and to then make a comparison of those COAs based on many factors including the situation, the commander's guidance, the principles of war, and the tenets of army operations. The commander makes the final decision on which COA will be used to generate his or her plan based on the recommendations of the staff and his or her own experience with the same factors considered by the staff (Alphatech, 1998).

The Disciple critiquer will identify strengths and weaknesses of a course of action with respect to the principles of war and the tenets of army operations (FM-105, 1993). There are nine principles of war: objective, offensive, mass, economy of force, maneuver, unity of command, security, surprise, and simplicity. They provide

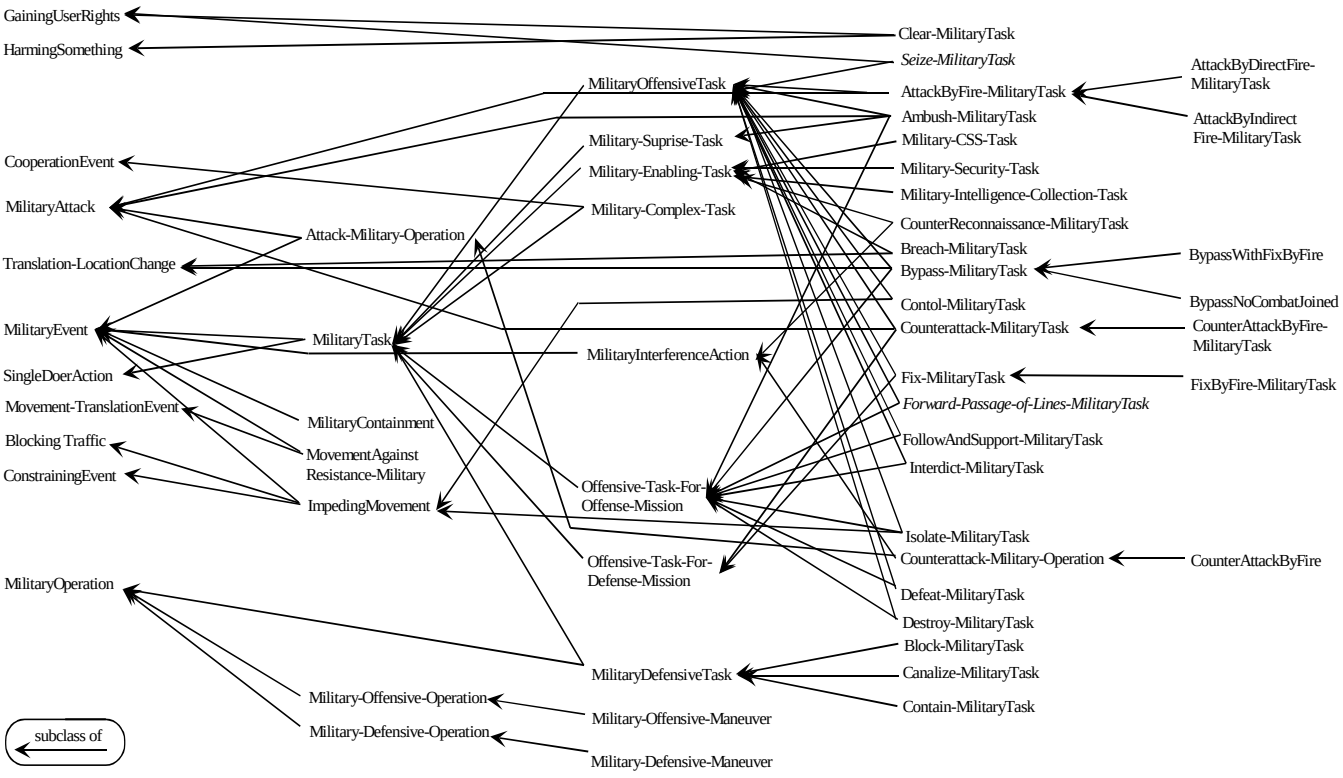

Figure 1: A fragment of the military tasks ontology.

general guidance for the conduct of war at the strategic, operational and tactical levels. The tenets of army operations describe the characteristics of successful operations. They are: initiative, agility, depth, synchronization and versatility.

The Disciple agent is developed to act as an assistant to a military commander and his or her staff, helping them to choose the best course of action for a particular mission. To develop the agent's knowledge base, we are performing a task based modeling of the principles of war and tenets of army operations. By task based, we mean the consideration of military units and the tasks they are assigned. We specifically look at:

- What tasks are assigned to units of interest?
- Can units accomplish assigned tasks in terms of the tasks being appropriate for the unit type, size and or condition?
- Is completion of the assigned tasks likely to contribute to the success of the overall mission?

In the next section we will present the knowledge representation used by the critiquing agent, giving examples from this domain.

Disciple Knowledge Base = Ontology + Rules

The problem solving approach of Disciple is based on the 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: objects, features, tasks, examples, explanations, and rules. The objects, features and tasks form the agent's ontology, and are the basic knowledge elements used to represent the problem solving rules. The examples and the explanations are temporary knowledge elements used to learn the rules.

In the following we will briefly present each of these elements.

Objects

The objects represent either specific individuals or sets of individuals in the application domain. The objects are hierarchically organized according to the generalization relation (subclass-of/superclass-of and instance-of/type-of). Figure 1, for instance, presents a fragment of the military task ontology used to model the COA analysis domain. As one can see, an object may have more than one parent. The hierarchy of objects is used by the Disciple agent as a generalization hierarchy, one way to generalize an expression being that of replacing an object with a more general one from such a hierarchy. Obviously, there are several ways in which an object could be generalized, selecting the right one being an objective of the learning process.

Features

The features and the sets of features 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. Expressions are generalized or specialized by adding or deleting features of the objects appearing in their descriptions. In the current version of Disciple, the values of the features may be generalized or specialized, but the features themselves are not generalized. However, the feature generalization hierarchies are extensively used in analogical reasoning.

Tasks

A task is a representation of anything that the agent may be asked to accomplish. The tasks and the sets of tasks are also hierarchically organized, according to the more general than relation. Figure 2, for instance, represents a fragment from the task hierarchy of the COA domain. The hierarchies of tasks are used in analogical reasoning. One should not confuse the military tasks presented in Figure 1 (which are represented as Disciple objects) with the agent's reasoning tasks which are represented as Disciple tasks.

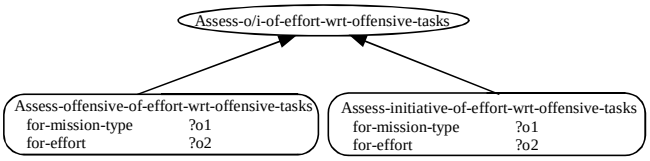

Figure 2: A fragment of the Disciple task hierarchy.

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

Examples

An example is a reduction of a specific task into subtasks:

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

A task may be reduced to one simpler task, or to a (partially ordered) set of tasks. The following is an example of task reduction:

IF ***the task to accomplish is***

Assess-initiative-of-effort-wrt-offensive-tasks
for-mission-type "military-offensive-operation"
for-effort armor-brigade1

THEN ***accomplish the task***

Assess-initiative-of-effort-with-two-tasks-wrt-offensive-tasks
for-mission-type "military-offensive-operation"
for-effort armor-brigade1
for-task1 forward-passage1
for-task2 seize1

Figure 3. An example of task reduction

Correct task reductions are called positive examples and incorrect ones are called negative examples.

Explanations

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 subject matter expert to a specific task reduction:

the task reduction TR is correct because E

For instance, the explanation of the task in Figure 3 is presented in Figure 4. It basically states that in order to assess the initiative of armor-brigade1 with respect to offensive tasks one has to assess the initiative of the two tasks forward-passage1 and seize1 because armor-brigade1 has two tasks and these tasks are forward-passage1 and seize1.

```
armor-brigade1   number-tasks   2
armor-brigade1   unit-mission   action1-of-armor-brigade1
                                   has-task   forward-passage1
armor-brigade1   unit-mission   action2-of-armor-brigade1
                                   has-task   seize1
```

Figure 4: The explanation of the example in Figure 3.

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

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

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

However, this example can also be represented in the equivalent form:

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

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

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

Task-reduction rules

The task reduction rules are generalizations of specific reductions, such as (1), and are learned by the agent through an interaction with the subject matter expert, as described in (Tecuci, 1998):

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

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

An 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 $T_{11g}, \ldots, T_{1ng}$

```
IF the task to accomplish is
   Assess-initiative-of-effort-wrt-offensive-tasks
         for-mission-type    ?o1
         for-effort          ?o2

   plausible upper bound
   ?o1    is        {"military-offensive-operation",
                     "military-defensive-operation"}
   ?o2    is        modern-military-unit
         number-tasks   [2 10]
         unit-mission   ?o5, ?o6
   ?o3    is        military-task
   ?o4    is        military-task
   ?o5    is        military-action
         has-task  ?o3
   ?o6    is        military-action
         has-task  ?o4

   plausible upper bound
   ?o1    is        {"military-offensive-operation"}
   ?o2    is        armor-brigade1
         number-tasks   [2 2]
         unit-mission   ?o5, ?o6
   ?o3    is        forward-passage1
   ?o4    is        seize1
   ?o5    is        action1-of-armor-brigade1
         has-task  ?o3
   ?o6    is        action2-of-armor-brigade1
         has-task  ?o4

THEN accomplish the task
   Assess-initiative-of-effort-with-two-tasks-wrt-offensive-tasks
         for-mission-type    ?o1
         for-effort          ?o2
         for-task1           ?o3
         for-task2           ?o4
```

Figure 5: A learned PVS rule.

We will refer to such a rule as a plausible version space rule, or PVS rule. Plausible version space rules are used in problem solving to generate task reductions with different degrees of plausibility, depending on which of its conditions are satisfied.

If the PLB condition is satisfied, then the reduction is very likely to be correct. If PLB is not satisfied, but PUB is satisfied, then the solution is considered only plausible. The same rule could also be applied for tasks that are considered similar to T_{1g}. In such a case the reductions are considered even less plausible.

Any application of a PVS rule however, either successful or not, provides an additional (positive or negative) example, and possibly an additional explanation, that are used by the agent to further improve the rule.

For instance, the rule learned by Disciple-LAS from the example in Figure 3 and its explanation in Figure 4 is presented in Figure 5.

Domain modeling

The process of building the knowledge base consists of three main steps: domain modeling, ontology creation and rule learning.

First 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.

Figure 6 illustrates how one can model the process of answering a question about a military course of action as a task reduction process. First the question

> "To what extent does this course of action embody the principle of objective?"

is expressed as a task to perform:

> "Assess the course of action from the point of view of objective."

Then this assessment task is successively reduced to simpler assessment tasks and ultimately reduced to assertions on how the course of action conforms to the principle of objective. From an ontology development point of view, this process is important because it informally identifies the concepts that are needed to be present in the ontology, as will be explained in the following. As one can see from Figure 6, to "Assess the course of action from the point of view of objective", one has to consider the features that characterize the objective, and these are specified by the subject matter expert as being "identification", "attainability", and "decisiveness". Therefore, the current task is reduced to the following simpler assessment tasks:

> "Assess identification of objective"
> "Assess attainability of objective"
> "Assess decisiveness of objective"

Assessing the attainability of the objective is most applicable for the main effort and an offensive mission. Therefore, the ontology has to contain a classification of COA missions into offensive missions and defensive missions, and each specific course of action would need to

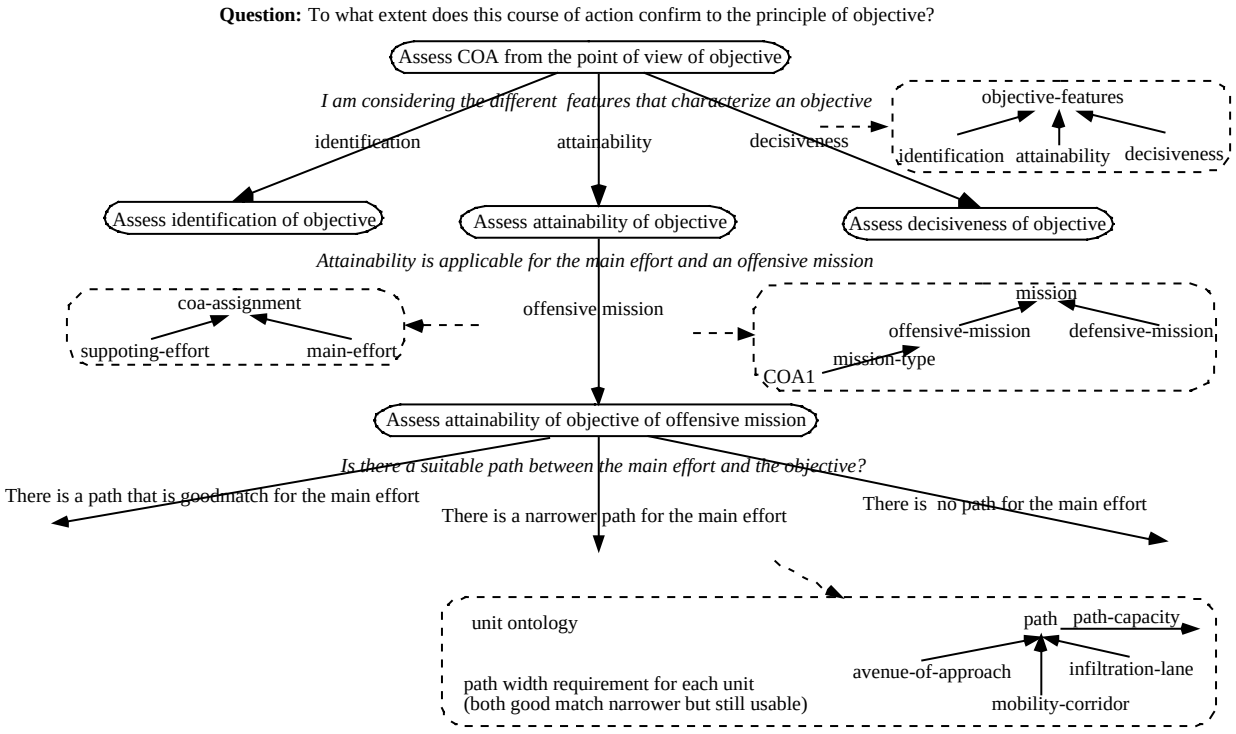

Figure 6: Conceptual task reductions

have a feature that will specify its mission type. To effectively assess the attainability of the objective one would have to determine whether there is a suitable path between the main effort and the objective. This reveals other necessary concepts, such as different types of paths (avenues of approach, mobility corridors, and infiltration lanes), and different types of units. Moreover, each path would need to be characterized by its capacity, and each unit would need to be described in terms of its path requirements. This process continues in this manner, the subject matter expert and the knowledge engineer defining the conceptual way of answering the various questions posed to the system.

There are several important results of this modeling process. First, an informal description of the agent's reasoning tasks is produced. These tasks will be latter defined more formally in terms of their features. Secondly, necessary concepts and features are identified, forming a basis for the ontology creation. Thirdly, the conceptual task reductions themselves are maintained to guide the training of the agent by the subject matter expert. As a result of this training the agent will learn general task reduction rules.

As one can see, the elements that need to be represented in the ontology result from an analysis of the agent's task reduction process in a particular application domain. This is the reason we call the process of building the ontology a domain-oriented one.

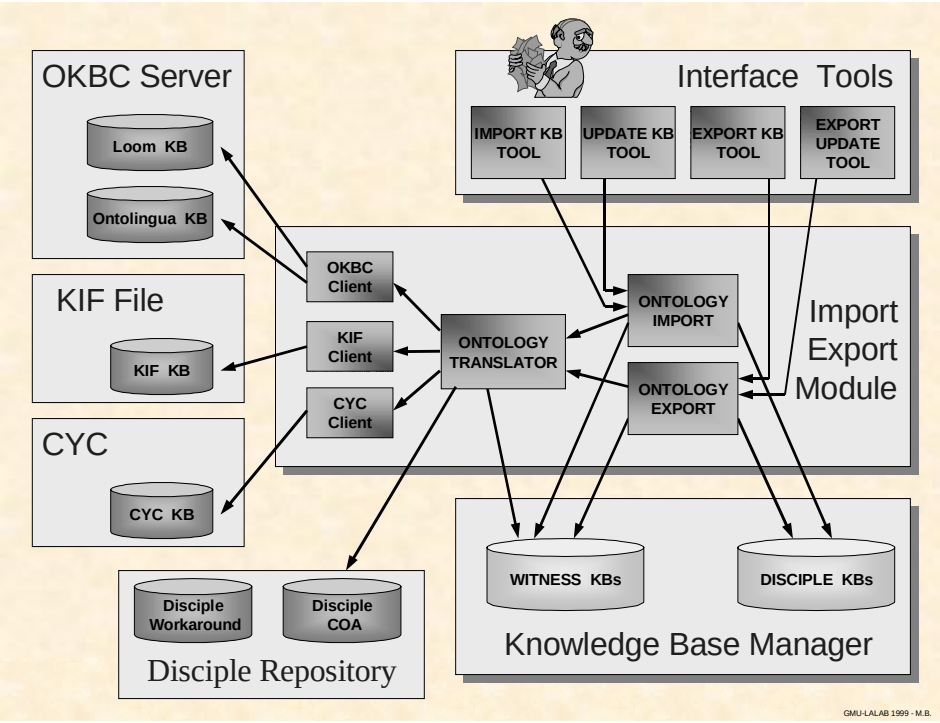

Figure 7: The architecture of the Import/Export module.

Knowledge Import and Export

As a result of the modeling process, a significant number of necessary concepts and features have been identified. Interacting with the Knowledge Import/Export Module, the subject matter expert and the knowledge engineer will attempt to import formal descriptions of these concepts and features from existing repositories of knowledge.

Figure 7 presents the proposed architecture of the Import/Export module of Disciple. Although only a simplified version of this architecture is currently implemented, we will use it to present our approach to the knowledge reuse problem.

We reduce the problem of importing knowledge from an outside knowledge server into two simpler problems: a translation problem and an integration problem. Let External-KB be an external knowledge base from which we want to import knowledge. External-KB might be, for instance, an Ontolingua KB (Farquhar et al. 1996), a Loom KB (MacGregor 1991), a CYC KB (Lenat 1995), or even a text file containing a KIF representation of any KB. Let Disciple-KB be the knowledge base that is under development with Disciple. We first translate External-KB into a Disciple KB, called Witness-KB, that can be browsed and edited using the Disciple ontology tools. Using these tools, the subject matter expert and the knowledge engineer can easily transfer knowledge elements from the Witness-KB into the Disciple-KB. In the same time, the knowledge elements imported in Disciple-KB are correspondingly marked both in the Witness-KB and in the Disciple-KB. If the External-KB has evolved to incorporate new knowledge elements, then one can re-import knowledge from it. In this case, the additional knowledge content of the External-KB is added to the Witness-KB and is marked accordingly. Therefore, one can easily see in the Witness-KB which are the new knowledge elements that may need to be incorporated into Disciple-KB.

Knowledge export involves the same processes in the reverse direction. In this case we look for the differences between the Disciple-KB and the Witness-KB and generate a new KB with these differences. This KB can then be used by the managers of the External-KB to update their KB.

Figure 7 shows the various Disciple interfaces and modules that will support the knowledge import/export process.

There are several strengths and weaknesses of this approach. One strength is that the translation process can be automatic and once we have translated an external KB into a Disciple KB we are within the Disciple environment.

In this environment, the knowledge engineer and the subject matter expert can use the specialized Disciple tool to extract the relevant knowledge from the Witness-KB. This means that they do not need to be familiar with tools other than those of Disciple, to be able to import knowledge from practically any external KB. Another strength is that we can develop advanced knowledge import tools that are independent of the change of the external knowledge repositories. For instance, such an advanced tool would take as input the descriptions of the concepts identified during the domain modeling process and will automatically browse the Witness-KB (which is the translation of the External-KB) in search for matching knowledge pieces, by using natural language based flexible matching strategies. The identified knowledge pieces can then be analyzed by the subject matter expert and the knowledge engineer and the relevant ones can be integrated into the knowledge base of the Disciple agent. This would relieve the Disciple knowledge base developers from manually searching a large and unfamiliar witness-KB, in search of the knowledge elements to be imported.

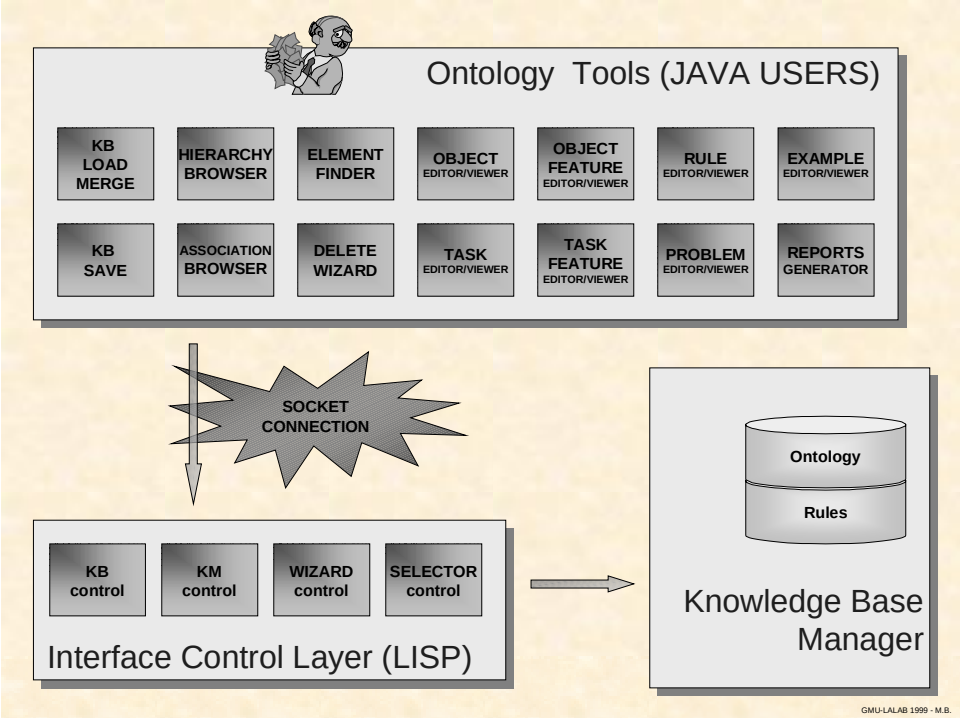

Figure 8: Ontology management tools

One weakness of the proposed approach is that, through automatic translation into the Disciple representation, relevant information might be lost, or represented in a form that can not be easily understood. This is a valid concern, especially when the external KB uses a knowledge representation that is more powerful than that of Disciple. On the other hand, we are really concerned here with the import of the ontological knowledge into a knowledge representation compatible with the OKBC knowledge model. Therefore, if we are importing from a frame-based system, the translation should not be a difficult problem. We are not concerned with the translation of the axioms (which would be a more difficult process) because we consider the axioms to be less reusable, and because Disciple is a tool specially developed to easily learn such axioms or rules from the subject matter expert.

Figure 7 shows that the external KB can also be a previously developed Disciple KB. In such a case, the translation process should also be trivial or unnecessary.

A simplified version of the above procedure for knowledge import is currently used to import ontological knowledge from CYC. First CYC ontological knowledge was translated into a KIF text file. Then the KIF file was automatically translated into a Disciple KB which was used as the initial KB for our COA critiquer. This KB was further extended manually with new knowledge pieces.

We have also imported knowledge from the LOOM server (MacGregor, 1991) using the OKBC protocol, when we have developed the knowledge base of the Disciple workaround generator (Cohen et al., 1998). In particular, we have imported 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.

Ontology building tools

Figure 8 presents the ontology management tools of Disciple. These tools are in various stages of developments. We have defined a specialized editor for each type of knowledge element to facilitate the interaction with the users. For instance, there is an object editor, a feature editor, a task editor, an example editor, and a rule editor. We are developing these tools to be user-friendly and uniform in both their appearance and operation. An important aspect of Disciple is the development of tools that are dedicated to the type of the user. In Disciple we distinguish between a knowledge engineer, a subject matter expert, and an end-user. Each tool will have different characteristics, depending of the user type. For instance, a tool for the knowledge engineer has many options and allows total control to its user. On the contrary, a tool for

an end-user will have only a few options and will provide close guidance.

Because the basic functions of these tools are indicated by their names, we will only discuss of few of them. The KB save and load tools allow the KB developer to split a knowledge base into various components (such as rules, objects, instances, features, or tasks) and save these components into separate knowledge bases. Then new knowledge bases can be assembled from these components.

The association browser manages the relations between the various knowledge pieces in the KB. It allows, for instance, to view all the objects that have a certain feature in their definition, or all the rules that have a certain concept in their definition.

We attempt to provide each tool with a certain degree of "intelligence", based on "wizard" control mechanisms that try to anticipate the most desirable parameter settings in a tool and to pre-select them when the tool is invoked.

An important ontology management tool is the delete wizard. This wizard is automatically invoked when the user attempts to delete an element from the KB. It guides the user through a sequence of modifications of the KB that are necessary in order to maintain the consistency of the KB. Let us consider, for instance, that the user wishes to delete an object O from the KB. Many other elements from the KB may be affected by this operation. Among them are the objects that are subconcepts (subclasses) or instances (individuals) of O, the objects that have a feature which value is O, the features that have a domain or a range that includes O, the rules that refer to O, and others. Therefore, before deleting O, one has to make sure that the definitions of all these elements are updated to no longer refer to O. But this is not a trivial task. Therefore the delete wizard analyzes the element to be deleted and generates a plan of actions to be performed by the user. Then it guides and helps the user to perform each step of the plan. For instance, when the wizard treats the objects that are subconcepts (subclasses) of O the user has to decide which will be their new parents, or whether they should also be deleted (and, of course, this would cause the whole process to be recursively applied). The user should also decide how to update the features that were inherited from O, and so on. The deletion of O affects also not only the rules that contain O in their conditions, but also the rules where O is between the plausible upper bound and the plausible lower bound. Therefore, the delete wizard is critical to the management of the KB.

Teaching the Disciple Agent

Once the ontology has been created, the expert can start to teach Disciple to solve problems in a cooperative, step by step, problem-solving scenario. In this process the expert will be guided by the conceptual task reductions that have been previously defined (see Figure 2). This teaching methods are described in detail in (Tecuci, 1998). Figure 9 presents the main processes of knowledge acquisition and learning that take place during the teaching of the agent.

During *Rule Learning*, the expert teaches the agent how

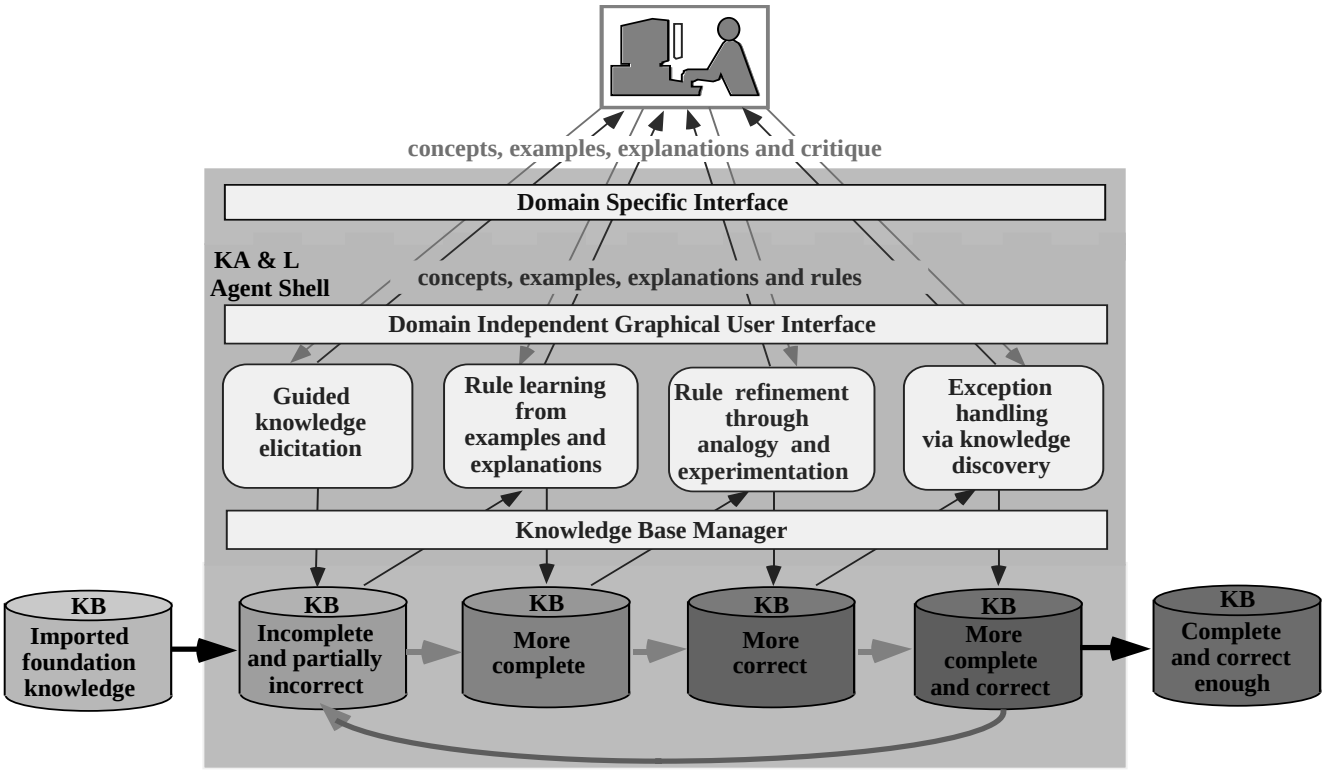

Figure 9: The main processes of knowledge acquisition and learning in Disciple.

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. Rule refinement will also cause a refinement of the concepts from the agent's ontology.

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.

Therefore, an important aspect of the teaching process is that it may also trigger an extension of the ontology.

The Knowledge Base Manager

Disciple-LAS is developed in a university research environment that imposes several important constraints on the design of its tools, including the ontology management ones.

One constraint is that the system should allow an extension of its knowledge representation, to cope with the challenges of new research projects. For instance, over the last two years, we had to extend several times the knowledge representation of Disciple. First we have adapted the representation of the Disciple ontology to be compatible with a subset of the OKBC knowledge model, in order to facilitate knowledge sharing based on the OKBC protocol. Then we have added the capability to compute arbitrary functions to the Disciple rules, in order to develop the workaroud agent and its knowledge base that required many numerical computations. Then we have extended the knowledge model of Disciple to include hierarchies of features and tasks, in order to enhance its analogical reasoning capabilities.

We also wanted to develop a prototype system that can be enhanced to deal with more complex application domains and increased requirements for speed and memory.

We have addressed these issues through a special design of the knowledge base manager, that combines a flexible multi-layer structure with an efficient KB access and management. The architecture of the KB manager is presented in Figure 10.

The KB manager has a hierarchical structure consisting of three layers: an upper knowledge management layer, a lower storage layer, and a link layer in between.

The KM Layer

The KM Layer implements specific macros and functions for each type of knowledge base element (object, feature, task, rule, etc.). All the upper modules of Disciple-LAS access the KB through these functions. Therefore, an extension in the knowledge representation will only require a modification of the macros and the functions of the KM layer, without affecting the upper modules of Disciple-LAS.

An important feature of the KM Layer is the management of special system-level properties for the elements of the Disciple hierarchies (such as "level' that indicates the height of an element in a hierarchy, and 'mark' that traces the elements of the KB that have already been scanned in a given search operation). They provide an infrastructure for fast learning and problem solving algorithms, and allow complex operations, such as the generalization or the specialization of concepts, to be performed in polynomial time.

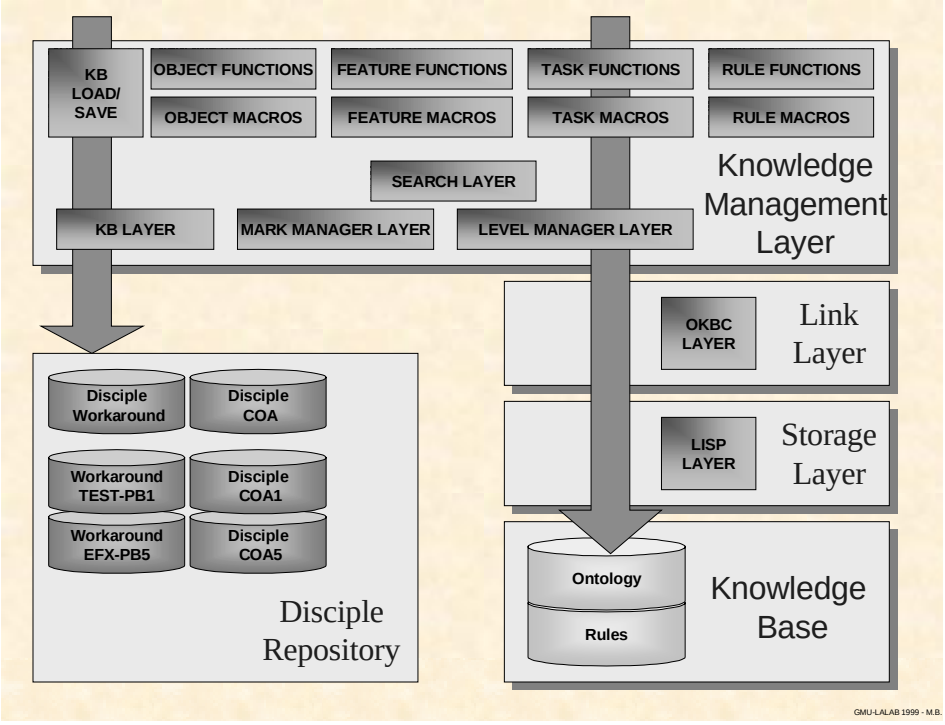

Figure 10: The architecture of the KB Manager.

The Link Layer

The KM Layer is implemented by using a subset of the OKBC functions for frame/slot representations. The Link Layer acts as a buffer between the KM Layer and the Storage Layer so that changes in the storage layer will only affect the implementation of the basic OKBC functions used.

The Storage Layer

The Storage Layer assures the low-level management of the KB elements in the storage environment. Currently, the storage layer of Disciple is represented by the LISP symbol layer, where the KB elements are represented as LISP symbols and their properties. This means that the size of the KB is limited by the LISP memory. However, as mentioned in the previous section, one can replace the LISP symbol layer with an efficient memory management system, such as PARKA (Stoffel et al. 1997), by re-implementing the OKBC access functions used in the link layer in the PARKA query language.

A feature that contributes significantly to the efficiency of the knowledge management operations is the extensive use of the macro operations, both in the KM Layer and in the Link Layer so that, even if the KB manager consists of three layers, invoking a KM operation is pre-proceed into a storage layer function.

The continuous evolution of the knowledge representation of Disciple-LAS, as well as its efficiency in problem solving and learning, support the claim that the Disciple KB Manager succeeds in achieving two apparently contradictory goals: flexibility and efficiency.

Conclusion

We have presented the domain-oriented approach to ontology creation and management that is part of the Disciple methodology for rapid development of knowledge-based agents. A characteristic feature of this approach is that a detailed specification of the ontology to be developed results from a conceptual modeling of the application domain of the knowledge-based agent to be built. Moreover, the employment of the OKBC knowledge model in Disciple facilitates the import of the identified concepts from OKBC knowledge servers. This ontology creation and maintenance approach is applied to the development of a knowledge base for COA analysis.

Future research will consist in extending this approach and the supporting tools to allow several experts and knowledge engineers to collaborate in building different parts of a knowledge base. We will further develop the knowledge import/export methods and module, and will investigate the creation of a reusable repository of Disciple knowledge bases. We will continue our work on the development of special interface tools to be used directly by the subject matter experts, as well as of wizards that will assist the experts. We will also investigate the creation of tools for supporting the domain modeling process.

Acknowledgments. This research is supported by the AFOSR grant F49620-97-1-0188, as part of the DARPA's High Performance Knowledge Bases Program.

References

Alphatech, Inc. 1998. *HPKB Course of Action Challenge Problem Specification*, Burlington, MA, December 2nd.

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.

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.

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.

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

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

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

FM-105. 1993. US Army Field Manual 100-5, Operations, Headquarters, Department of the Army, June 1993.