The Shifting Terminological Space: An Impediment to Evolvability

William Swartout
Robert Neches
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

Abstract

In an expert system, rules or methods interact by creating situations to which other rules or methods respond. We call the language in which these situations are represented the *terminological space*. In most expert systems, terms in this language often lack an independent definition, in which case they are implicitly defined by the way the rules or methods react to them. We argue that this hampers evolution, and argue for a separate, independently defined terminological space that is automatically maintained.

1. Introduction

Due to the experiential, incremental nature of expert system development, it is critical that an expert system shell support the addition, modification and deletion of knowledge with minimal perturbation of the rest of the knowledge base and that a developer be able to determine how a proposed change will affect the behavior of the system.

What's needed for evolvability? There seem to be two major factors. First, the expert system organization should be *modular* in the sense that it should be possible to change one part of the system independently of the rest of the system. Second, the expert system organization should be *explicit* so that the effect of a change can be readily understood.

Advocates of rule-based expert system shells argue that this desired modularity arises inherently from the use of rules [6]. However, practical experience indicates that merely adopting a rule-based framework does not guarantee modularity and in some ways can impede it [3]. In this paper we will outline some of the more subtle factors that seem to affect evolvability. Focusing on one that we call the *shifting terminological space*, we will describe how we have addressed these issues in our the Explainable Expert Systems (EES) framework [11]. The EES project is exploring a new paradigm of expert system development in which the role of knowledge engineers and domain experts is to develop a rich and detailed knowledge base that captures the factual knowledge and problem solving methods in a domain. Executable code for the expert system is then derived from the knowledge representation. Systems built in this fashion are expected to have a richer knowledge base from which to support machine-generated English explanations and justifications, as well as a more modular structure and principled organization that will facilitate the development and maintenance process.

2. Impediments to Evolvability

We begin by reviewing characteristics of current expert system frameworks that limit the modularity of expert systems.

An overly specific representation for knowledge. Knowledge is stated at a low level and is very specific to a particular task [11]. For example, MYCIN is sometimes able to determine the genus of the micro-organism infecting a patient but unable to determine its species. When this occurs, MYCIN just assumes that the species of the organism is the most likely one for the particular genus. This is a reasonable default rule, but unfortunately the general heuristic is not represented at all in MYCIN. Instead, it is captured by a set of rules, each one specific to one of the genera MYCIN knows about. For system evolvability this overly specific representation of knowledge is a major problem. From the standpoint of modularity, if one wanted to modify the general heuristic MYCIN employed, there is no single rule to modify. Instead, the system builder would have to locate and modify manually each of the rules that instantiated the general heuristic, with all the attendant possibilities for making a mistake. MYCIN also reduces explicitness, by forcing the system builder to express knowledge at an overly specific level that does not say what the system builder really intended to represent.

Confounding of different kinds of knowledge. As Clancey [3] has pointed out, a single rule may combine a variety of different kinds of knowledge, such as domain facts, problem-solving knowledge, and terminology. This reduces the modularity of an expert system because the different kinds of knowledge cannot be modified independently. This compilation of knowledge also occurs, unrecorded, in the head of the system builder, reducing explicitness and tending to make machine-provided explanations difficult to understand.

Impoverished control structures. Rules do not cleanly support some higher-level control structures, such as iteration [4, 13]. On those occasions when it is necessary to represent an iterative control structure within a rule-based architecture, the results are usually clumsy and introduce serious interdependencies among the rules. Modularity is reduced because of the interdependencies. Explicitness is also reduced because the interdependencies are artifacts of the architecture and not what the system builder really wanted to say.

Limited (or no) representation of intent. In most expert systems, rules are written at what Davis calls the "object level" [5], that is, at the level of domain phenomenon. For example, as Clancey points out [3], MYCIN's higher-level strategies are implicitly encoded in the rules. There is no representation of how particular rules are involved in achieving these strategies, and in fact, the strategies aren't even represented. Instead, strategies are carefully (and implicitly) encoded by mapping intentions to object level terms. For instance, MYCIN lacks any explicit representation of the diagnostic strategy it employs; instead, that strategy has been carefully encoded by the system builder in MYCIN's rules and expressed at the object level. This mapping

hampers both explicitness and modularity because the mapping usually does not remain valid as the system grows.

To the four problems above, we add a fifth:

An ill-defined terminological space. Laird and Newell define the *problem state* as "the current state of knowledge of the task situation [that] exists in the agent in some representation" [9]. Rules interact by creating situations or problem states that will trigger other rules. Thus, the language in which the problem state is expressed (which we call the *terminological space*) is the "glue" that holds the whole expert system together. Unfortunately, some (if not most) expert systems lack any independent definition of terminology. Instead, the terms asserted by a rule acquire their meaning based solely on how other rules in the system will react to them. This is undesirable because the definition of terms is implicit; it can also seriously affect a system's modularity, since it means that changing a rule can affect not only the behavior of that rule, but also the meaning of any terms that it employs. This, in turn, can implicitly affect the behavior of any rules that use or assert that term.

We have described elsewhere how we have dealt with the first four problems in the EES framework (see [11] for more details). This paper will focus on our solution to the fifth problem: how we have chosen to represent the terminological state and how we express knowledge within that space.

To deal with the problem of an ill-defined terminological space, we decided to adopt NIKL [10], a descendant of KL-ONE [2], as our basic knowledge representation language. NIKL is a semantic network-based formalism and was designed to provide a semantics for concepts that is independent of how those concepts are used in a particular system. Thus, the way a concept is defined within NIKL itself gives it its meaning, rather than having the concept acquire its meaning based on how it is used within the expert system. Because NIKL provides an independent semantics for concepts, its designers were able to provide an automatic classifier [12] for NIKL. This classifier can determine subsumption relations among NIKL concepts based solely on their definitions.

In the remainder of this report we will describe how we have represented intent and the terminological space within EES, and how we have made use of NIKL: what seems to be good about this sort of principled knowledge representation language, where the problems are, and what opportunities and challenges remain.

3. Representing Intent in a Terminological Space

Representing intent explicitly means providing an explicit representation for the goals the system is trying to achieve and the means for achieving those goals. As we have argued above, many expert systems are written at an object level and lack an explicit representation of the goals they are attacking. The lack of an explicit representation for goals reduces a system's understandability and its modularity.

However, it is insufficient just to provide a representation for goals: how the goals acquire their meaning is also critical. Because this problem is generally not recognized, many systems let the goals acquire their meanings based on how those goals are achieved. This means, for example, that a *plan* for performing a diagnosis implicitly defines what performing a diagnosis means. The problem with this approach is that it confuses *how* the goal is achieved (that is, the plan) with *what* it means to achieve the goal. Following this approach can lead to real problems in evolvability because changing the way a goal is achieved implicitly changes the meaning of that goal.

A possible objection to requiring that goals be represented explicitly is that it means that the system reacts to goals that have been posted rather than to object-level data. Thus, the system is not free to opportunistically react to novel situations, but rather is more tightly controlled and in a sense has to know what it's doing. The ability of a system to opportunistically react to novel situations in ways unanticipated by the system builder is, of course, one of the oft-touted advantages of a rule-based expert system architecture, but it seems to work more in theory than in practice. The problem is that it is difficult to specify the condition part of a rule so that the action part is performed in just the right circumstances. The result is that the reaction of the system in unanticipated situations is likely to be wrong, and in fact, a large part of developing a rule-based expert system involves fixing the rule-base so the system does the right thing in unanticipated situations. Our approach is more conservative in that it gives us tighter control over both the situations our systems will handle and the results they will produce.

In EES, we have partially addressed the need for independent and explicit goal definitions by representing goals as NIKL concepts. The goals thus acquire their meaning from the semantics of NIKL, rather than from the methods that are used to achieve them. Although we feel this is the right approach to take, we feel we have only partially addressed the problem because NIKL does not allow us to represent everything we would like to be able to represent about a goal.

For our purposes, NIKL can be summarized as having a small set of basic capabilities. (The true set of capabilities, although still manageably small from the standpoint of maintaining fully-defined semantics, is somewhat broader than the simplified picture presented here. For more details, see [12].) New NIKL concepts are created from other concepts through modification of or addition to the roles associated with a concept. Roles in NIKL correspond roughly to slots in frame-based representation languages. When a role is associated with a concept, a *value restriction* may also be given that specifies possible fillers for that role at that concept. *Number restrictions* are also associated with roles at concepts and specify how many objects may fill that role at that concept. Specializations of a concept are created either by further restricting an existing role on a concept, or by adding an additional role. Roughly, a concept A subsumes a concept B if all the roles on A are present on B and the number and value restrictions on B are at least as tight as on A[1].

[1] This simplified view ignores some details having to do with primitive concepts see [12]

Each goal represented in an EES knowledge base has three major NIKL *roles*: a *requirements description*, *inputs*, and *outputs*. We can represent this as follows: ***GOAL = [OBJECT: requirement-description ACTION, input (arbitrary) OBJECT, output (arbitrary)OBJECT]***.[2]

The filler of the requirement description role is a NIKL concept which is subsumed by the NIKL concept *action*. It represents the intention of the goal, that is, what the goal is intended to accomplish. For example, English paraphrases of the requirements of some of the goals we have modelled (from several different domains) are: "compensate digitalis dose for digitalis sensitivities", "locate cause of fault within ground-system" and "scan program for opportunity to apply readability-enhancing transformations". An important point is that these requirements are not atomic, but are themselves structured NIKL concepts composed of more primitive concepts. The structured nature of the requirements is important because it eases the process of producing natural language paraphrases and because it allows the classifier to infer subsumption relations among the concepts, something that would have to be manually inserted if the concepts were atomic.

The requirements description can be thought of as specifying the problem to be solved. The inputs and outputs of a goal specify respectively the data that is available in solving the problem (inputs) and the data that is expected to be returned as part of the solution (outputs).

Plans in EES are also represented as NIKL concepts, in a similar way to goals: ***PLAN = [OBJECT: capability-description ACTION, inputs (arbitrary) OBJECT, outputs (arbitrary) OBJECT, method <sequence of goals>]***.

The capability description of a plan describes what it can do. The inputs and outputs describe the data it expects to receive and provide respectively. The method is a sequence of subgoals that accomplish the action referred to in the capability description.

Automatic classification is important in the context of EES because we use subsumption relations in finding plans that can achieve goals. When a new concept is specified information is given about restrictions on its roles. The NIKL classifier can reason about those restrictions to decide about the actual placement of the concept in the hierarchy. It may find that the user-supplied initial description of the object places it higher than it actually belongs. It may find that the concept is subsumed by additional concepts that were not stated in the initial description. It may also find that the new concept has restrictions that make it more general than existing concepts, and interpose the new concept between those concepts and their former parents. For an example designed to illustrate many of the capabilities of the classifier, see [11]. The NIKL classifier automatically maintains subsumption relations, so it is possible for the system to "discover" plans that can be applied to a particular goal without the need for the system builder to anticipate that case in advance. The classifier provides an independent criteria for the meaning of goals and plans.

The process by which the program writer finds candidate plans for achieving a goal is as follows. When a goal is posted, the system retrieves the action represented as the goal's requirement-description. It then examines all actions which subsume that action, to see if any of them are the capability description of some plan. All such plans are checked to see if their inputs and outputs are compatible with the goal. Those which satisfy that constraint become candidate plans for achieving the goal. If several plans are found, they are tried in order, most specific first, until one succeeds.

We feel there are two ways this approach aids the evolvability of expert systems. First, intention is explicitly represented. We do not rely on the intention-to-object level mapping referred to earlier that is implicit in many expert systems. As a result, we don't have to worry about that mapping becoming invalid as the system expands. This makes the system more modular, and hence evolvable.

Second, since intention is expressed in terminology that has an independent semantics, the meanings of terms remains constant as the system's problem-solving knowledge expands or is modified. The meaning of the terms only changes if we choose to explicitly modify the terminological space itself. If it is necessary to do so, we can at least see what will be affected, because the NIKL knowledge base is cross-referenced so that all uses of a term can be found. This organization also increases modularity and evolvability.

3.1. Describing Actions: A Closer Look

Actions are the "glue" in our system that link goals and plans. In this section we look at them in detail. As we use actions in goals and plans, they are essentially descriptors of a transition the system builder intends to take place. We represent the action concepts as verb clauses. The main verb (e.g. "compensate", "locate", or "scan" in the above examples) has roles associated with it that correspond to the slots one would find in a case-frame representation of a verb clause in a case grammar. For example, the internal representation for the requirement "locate cause of fault within ground-system" is: ***LOCATE-1 = [LOCATE: obj [CAUSE of [FAULT within GROUND-SYSTEM]]]***.

This requirement is a specialization of the LOCATE concept, where the "obj" role (corresponding to an object slot in a case frame) is filled by "cause of fault within ground-system". We also represent prepositions that modify concepts as roles, so "of" and "within" are used to form specializations of the concepts CAUSE and FAULT in the above example.

3.2. An Example

Let's consider a simple example of the addition of new problem-solving knowledge from the domain of diagnosis of a space telemetry system. In this domain, we represent as domain descriptive knowledge[3] the structure of the telemetry system (i.e. the systems and subsystems and how things are interconnected.)

[2] Our notation for concepts is as follows: concepts appear in upper-case. Roles on concepts appear in lower-case. Value restrictions on the possible fillers of a role appear as concepts following the role. Number restrictions on the number of objects that may fill a role appear in parentheses preceding the value restriction. "arbitrary" indicates 0 to infinity fillers. If no number restriction is explicitly stated, the default is 1 or (number 1). A specialization of a concept is formed by further restricting or adding additional roles to an existing concept. In our notation, this is denoted by placing the concept to be specialized within brackets, followed by the modified or added roles. If the concept is to be given a name, that is denoted by an equal sign preceding the left bracket. Thus, in the example of the definition of GOAL above, we see that a GOAL is a specialization of OBJECT with a requirements-description role filled by an ACTION, and input and output roles filled by OBJECTs.

[3] *Domain descriptive knowledge* is the knowledge of the structure of the domain.

Problem-solving knowledge is a set of plans for diagnosing potential problems with the system. The terminological knowledge characterizes and (through the classifier) induces a classification hierarchy on the different types of systems, and provides a means for linking goals and plans. As an example, consider a very simple diagnostic strategy, which locates a fault within a component by considering each of its subcomponents in turn[4]. Naturally, such a plan will only work if the component actually has subcomponents. The capability description for such a plan would be a NIKL concept such as, ***[LOCATE: obj [CAUSE of [FAULT within DECOMPOSABLE-SYSTEM]]]***, where the concept DECOMPOSABLE-SYSTEM was defined as: ***DECOMPOSABLE-SYSTEM = [SYSTEM contains1 (minimum 1) SYSTEM]***.

That is, a DECOMPOSABLE-SYSTEM is one that contains a system[5]. Now, let's say that in the domain descriptive knowledge we describe one of the components of the system: ***SPACECRAFT = [SYSTEM: contains12 TRANSMITTER, contains13 SPACECRAFT-RECEIVER]***.

That is, a spacecraft is a system that contains two subcomponents: a transmitter and a spacecraft-receiver. The classifier will recognize that a spacecraft is a kind of decomposable-system, because it contains at least one system. Suppose a goal is posted whose requirements description is: ***[LOCATE: obj [CAUSE of [FAULT within SPACECRAFT]]]***. The system will find the plan above because the classifier recognizes that a spacecraft is a kind of decomposable-system.

Now, suppose we wanted to add a new problem-solving plan to the system that had specialized knowledge about how to locate faults in systems that contained transmitters. The capability description for this plan would be: ***[LOCATE obj [CAUSE of [FAULT within [SYSTEM contains47 TRANSMITTER]]]]***.

The terminological reasoning done by the classifier enables the system to determine that this plan is more specific. Therefore, the system will choose this plan in preference to the more general one for diagnosing decomposable systems, in those cases where it has to locate a fault in a system that contains a transmitter, such as "locate cause of fault within spacecraft". Thus, we can add new problem-solving knowledge and have the system apply it automatically in appropriate circumstances. This provides us with a very clean mechanism for embedding special-case knowledge within a system. Furthermore, such special case knowledge does not *replace* the old problem-solving knowledge when added to the system. Whenever possible, the system will try to apply special-case knowledge first because it is more specific, but the general knowledge is still available as a backup in case the special case knowledge fails.

Now, let's consider what happens when we have to modify the terminological space to add a new kind of system, and a new kind of problem-solving knowledge. Suppose our informant describes certain kinds of systems where the inputs and outputs among the subcomponents are so tightly-coupled that a different diagnosis strategy is more appropriate: e.g., signal-tracing within the system rather than the tree search method we use on decomposable systems. Unfortunately, as so often happens in expert system construction, our informant can't state precisely the defining characteristics of a tightly-coupled system, but can identify empirically which systems are tightly-coupled. Because we can't give a precise definition for tightly-coupled systems, we can't use the classifier to recognize which systems are tightly-coupled. Instead, we have to explicitly define systems as being tightly coupled. Thus, we define the concept "tightly-coupled-system" as a primitive specialization of system and, for example, explicitly define deep-space-receiver as a specialization of that: ***TIGHTLY-COUPLED-SYSTEM = [SYSTEM]***, and ***DEEP-SPACE-RECEIVER = [TIGHTLY-COUPLED-SYSTEM: contains23 ANTENNA-SYSTEM, contains24 GROUND-RECEIVER, contains25 SIGNAL-CONDITIONER]***.

As one would expect, the capability description for the new plan for diagnosis by signal-tracing would be: ***[LOCATE obj [CAUSE of [FAULT within.TIGHTLY-COUPLED-SYSTEM]]]***. The interesting thing is that when the goal to "locate cause of fault within deep-space-receiver" is posted, one of the plans that is found is "locate cause of fault within decomposable-system". This is because the classifier recognizes that a deep-space-receiver is a kind of decomposable-system. The system would try to apply the plan for tightly-coupled-systems first because it is more specific, but the plan for decomposable-systems would nevertheless still be available as a backup if the other plan failed.

We have been pleasantly surprised more than once by this sort of behavior. This example illustrates how the classifier, by automatically maintaining subsumption relations in the terminological space, allows the system to make appropriate use of its knowledge in ways that may not have been anticipated by the system builder.

3.3. Reformulation

An additional capability of the EES framework that makes use of NIKL and the terminological space is its ability to *reformulate* a goal into a new goal or set of goals when searching up the subsumption hierarchy fails to yield an acceptable plan for acheiving the original goal. This capability enhances evolvability by further loosening the coupling between plans and goals, thus making the system more modular. Explicitness is also enhanced, since the system performs -- and records -- the reformulation, rather than having a human system builder perform it and fail to note having done so. The reformulation capability is discussed in detail elsewhere [11, 14].

4. Limitations

Unfortunately, this approach is by no means without problems.

For example, there is no way to construct a concept that denotes "the thing that is the value restriction of the b role of A". Unfortunately, the need to do this arises frequently in our representation of method knowledge in plans. For example, if we decide to represent systems as having a role "component", and we have a particular system, "system1" that has a component "system2" there is no way in NIKL to represent something like "diagnose component of system1" and have NIKL recognize that that is equivalent to "diagnose system2". This would be very useful in representing abstract steps for a plan.

---

[4] We have constructed a demonstration system for this example, along with some of the other problem solving strategies in the domain. This system, while still small by expert system standards, begins to demonstrate the feasibility of this approach.

[5] In our implemented system, there is a *number restriction* placed on the contains role that indicates that there must be at least one system filling it. Also, "contains1" is a specialization of the "contains" role, necessary for reasons we won't go into here.

Another problem is that NIKL does not support transitive relations. This has made it difficult to represent some terms that involve causality, for example, and has forced us to handle some things with problem-solving knowledge that ideally should be handled by the classifier.

A third problem is that there are some conceptualizations where inherent ambiguities in the language make it impossible for the classifier to decide if one concept is subsumed under another without explicit guidance. For example, our diagnosis system has a concept of a "start of a component chain": a sub-component whose input is the same as the input of the component containing it. We might like the classifier to recognize when a new concept is subsumed under this concept. However, it is not enough for the classifier to simply define that new concept so that its input happens to be the same as that of its containing component. Doing so only tells NIKL that inputs of the concept and its containing component are subsumed under a common parent; for the classifier to recognize that the new concept is a specialization of "start of a component chain", we must explicitly indicate that the values of the two input roles are intended to be identical. This is an important limitation on the ability of the classifier to find relationships that the system builder failed to anticipate.

While limitations such as these have hampered our efforts to completely separate terminological knowledge from other kinds of knowledge, nevertheless we feel we have gained significant leverage from NIKL. Furthermore, NIKL is under continued development [7].

5. Status

The system has operated on reasonably large NIKL networks (approximately 500 concepts). The plan-finding and reformulation capabilities described in the preceding sections, as well as the NIKL knowledge base are all operational and support the examples given above.

6. Aids for Constructing Models

We have argued for a well-defined and independent terminological space and tried to show how that could ease the evolution of an expert system. However, the increased development time due to the demands of building the initial knowledge base is a severe potential problem. We are developing a knowledge acquisition aid to address this issue, which will help knowledge representation builders plan activities and keep track of status while developing knowledge bases. It does so by maintaining an agenda of unfinished business that the knowledge base builder must deal with before the specification can be considered complete. The aid seeks to ensure that consistent conventions are followed throughout a knowledge base in terms of both what kinds of information are represented and what form is followed in representing them.

Our knowledge acquisition problem is partly defined by our representational concerns. We have to be concerned with eliciting not just a sufficient set of some type of concept, but with forming an abstraction hierarchy and placing concepts properly within it. Furthermore, unlike acquisition aids like MORE [8] and ROGET [1] which essentially have a schema for a particular kind of system that they are trying to instantiate for a new domain, we expect the problem solving knowledge to be part of what is acquired. Thus, one cannot know in advance what kind of system is to be acquired, and therefore cannot make the same assumptions about what kinds of concepts need to be elicited.

The fundamental insight represented by the design of this knowledge acquisition aid is that the increased reliance on detailed representation of terminological knowledge creates an opportunity, and not just a problem. Because more concepts are explicitly represented, it becomes easier to communicate about how one intends to make use of them. In particular, we will allow the knowledge base builder to express intentions for the form and content of the evolving knowledge base. Those intentions can be stated in terms of relationships or predicates that are expected to hold for concepts that classify in some way with respect to existing concepts. We will then be able to check, as the knowledge base evolves, whether concepts have been specified that violate these expectations. These concepts then become the target for knowledge acquisition, as we try to bring them into line with the statements of intentions.

An example of a problem in knowledge base construction that this could help with has to do with two distinct uses of inheritance of roles in a taxonomic knowledge base. One use is to represent some information at the most abstract concept for which it holds; we expect that instances subsumed beneath that concept will simply inherit the information. Another use is to represent an abstraction of the information itself; in that case, we expect that instances of the concept will *not* inherit the associated abstract information but will instead indicate a more specific version of the information related by that role. Knowledge base errors can arise because there is ordinarily no information to indicate which of these two uses is intended. Thus a new item can appear to be complete because it has some role it was intended to have, when in fact the value of the role is an inherited value that is actually less specific than is needed. Take for example, the notion of program transformations in our the Program Enhancement Advisor. Building this system requires that the EES program writer reformulate a general goal to enhance programs into specializations like, ENHANCE READABILITY OF PROGRAM and ENHANCE EFFICIENCY OF PROGRAM. Implementing these goals, in turn, requires the program writer to instantiate a plan that calls (among other things) for a goal to SCAN PROGRAM FOR TRANSFORMATIONS THAT ENHANCE CHARACTERISTIC OF PROGRAM. Thus, a keystone concept in the model is: ***ENHANCE-1 = [ENHANCE: agent TRANSFORMATION, obj [CHARACTERISTIC of PROGRAM]]***.

In order to instantiate plans in the manner intended by the system-builders, there is an implicit requirement that the domain knowledge about transformations specifies, for each transformation, some *particular* CHARACTERISTIC of PROGRAMs which that transformation ENHANCEs. That is, to be useful to the Program Enhancement Advisor, a transformation has to be described as enhancing *something*. This is the knowledge that, in the end, determines what transformations the resulting expert system program will make use of in trying to suggest program enhancements. Thus, although many specializations of ENHANCE-1 are possible, certain specializations are crucial; the knowledge acquisition process must ensure that they are supplied. In particular, we expect to see specializations in which both the **agent** and the **obj** roles are further restricted, because we do not want inheritance to produce the spurious implication that all transformations enhance all characteristics of programs. In other words, the intention in specifying domain knowledge for this domain is to model *which* transformations enhance *which* characteristics.

The knowledge acquisition aid we are now designing is centered around a language for stating intentions such as these in terms of how various concepts are expected to classify. The aid will be able to look at the results of classification to determine whether or not such intentions have been violated. We can use these intentions to guide us in deciding whether further information is needed pertaining to a concept, and also when the knowledge base is or is not consistent with respect to its intended use (which is a different question from that of whether it is internally consistent).

7. Summary

In this paper we have argued that evolution is a critical part of the expert system lifecycle and that expert system frameworks must support evolvability. We have identified several factors that limit the evolvability of many expert systems and focussed on two: limited representation of intent and an ill-defined terminological space. Additionally, we have shown how such an independent terminological space could be used to define goals and plans. Finally, we have argued that a mechanism such as the NIKL classifier can be a significant benefit by finding and maintaining subsumption relations in the terminological space automatically, allowing the system to make use of knowledge in ways that may not have been foreseen by the system builder.

Acknowledgements

We would like to thank R. Balzer, R. Bates, L. Friedman, L. Johnson, T. Kaczmarek, T. Lipkis, W. Mark, J. Moore, J. Mostow, and S. Smoliar for interesting discussions and comments that greatly facilitated the work described here. The EES project was supported under DARPA Grant #MDA 903-81-C-0335.

References

1. Bennett, J., "ROGET: acquiring the conceptual structure of a diagnostic expert system," in *Proceedings of the IEEE Workshop on Principles of Knowledge-based Systems*, 1984.

2. Brachman, R. J., and Schmolze, J. G., "An Overview of the KL-ONE Knowledge Representation System," *Cognitive Science* 9, 1985, 171-216.

3. Clancey, W., "The Epistemology of a Rule-Based Expert System: A Framework for Explanation ," *Artificial Intelligence* 20, (3), 1983, 215-251.

4. Davis, R., *Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases*, Ph.D. thesis, Stanford University, 1976. also available as SAIL AIM-283

5. Davis, R. and Lenat D. B., *Knowledge-based systems in artificial intelligence*, McGraw-Hill, 1982.

6. Davis, R., King, J., *The Origin of Rule-Based Systems in AI*, Addison-Wesley, 1984.

7. Kaczmarek, T., Bates, R., and Robins, G., "Recent Developments in NIKL," in *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, 1986.

8. Kahn, G., Nowlan, S., McDermott, J., "A foundation for knowledge acquisition," in *Proceedings of the IEEE Workshop on Principles of Knowledge-based Systems*, 1984.

9. Laird, J. and Newell, A., *A Universal Weak Method*, Carnegie-Mellon University Department of Computer Science, Pittsburgh, PA, Technical Report CMU-CS-83-141, June 1983.

10. Moser, M.G., "An Overview of NIKL, the New Implementation of KL-ONE," in *Research in Natural Language Understanding*, Bolt, Beranek, & Newman, Inc., Cambridge, MA, 1983. BBN Technical Report 5421

11. Neches, R., W. Swartout, J. Moore, "Enhanced Maintenance and Explanation of Expert Systems through Explicit Models of Their Development," *Transactions On Software Engineering*, November 1985. Revised version of article in Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, December, 1984

12. Schmolze, J.G. & T.A. Lipkis, "Classification in the KL-ONE Knowledge Representation System," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, IJCAI, 1983.

13. Swartout, W., "A digitalis therapy advisor with explanations," in *Proceedings of the Fifth International Conference on Artificial Intelligence*, pp. 819-825, Cambridge, MA., 1977.

14. Swartout, W., "Beyond XPLAIN: toward more explainable expert systems;" in *Proceedings of the Congress of the American Association of Medical Systems and Informatics*, 1986.